Tatiana Kossaczká, Matthias Ehrhardt and Michael Günther

Deep FDM: Enhanced finite difference methods by deep learning

April 10, 2023

Deep FDM: Enhanced finite difference methods by deep learning

Tatiana Kossaczká*, Matthias Ehrhardt, Michael Günther

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM), Chair of Applied and Computational Mathematics, Bergische Universität Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany

Abstract

In this work, we propose a new idea to improve numerical methods for solving partial differential equations (PDEs) through a deep learning approach. The idea is based on an approximation of the local truncation error of the numerical method used to approximate the spatial derivatives of a given PDE. We present our idea as a proof of concept to improve the standard and compact finite difference methods (FDMs), but it can be easily generalized to other numerical methods.

Without losing the consistency and convergence of the FDM numerical scheme, we achieve a higher numerical accuracy in the presented one- and two-dimensional examples, even for parameter ranges outside the trained region. We also perform a time complexity analysis and show the efficiency of our method.

Keywords: Finite difference scheme, Compact finite difference scheme, Partial differential equations, Discretization error, Deep Learning, Deep Neural Networks
2000 MSC: 65M06, 68T07, 91G20

*corresponding author

Email addresses: kossaczka@uni-wuppertal.de (Tatiana Kossaczká), ehrhardt@uni-wuppertal.de (Matthias Ehrhardt), guenther@uni-wuppertal.de (Michael Günther)

1. Introduction

Many practical problems, e.g. in quantitative finance, stochastic control and quantum physics, can be modeled by partial differential equations (PDEs) which in most cases do not admit analytical solutions. Thus, it is inevitable to approximate the solutions of the PDEs by numerical methods, such as finite differences, finite elements, finite volumes, radial basis functions, etc. Besides stability issues, the user is also concerned with the efficiency of the numerical method, i.e. the relation of achieved accuracy to the required computation time.

In recent years, there has been an increased interest in solving PDEs using Deep Learning, see e.g. [1, 2, 3, 4, 5, 6, 7]. This interest was mainly due to the availability of new generations of computers and a major challenge that applies to all grid-based solution methods: the curse of dimensionality, which very often occurs e.g in portfolio optimization, where the spatial dimension corresponds to the number of assets. We refer the reader to [2] for further information on deep neural networks (DNNs) methods for solving PDEs in high-dimensions. On the other hand, DNN-based PDE solvers generally cannot compete with classical numerical solution techniques in lower dimensions - since solving the highly nonlinear optimization problems in the training phase is too costly.

In addition, neural networks have a compositional structure that provides new approximations for highly nonlinear functions, and that in some ways complements conventional linear, additive forms of basis functions, e.g., in finite element methods. However, exactly this flexibility of DNNs as a universal approximation method comes at the expense of a large number of parameters ('hyperparameters') that need to be determined during the supervised learning phase. Also, often machine learning based solver still lack mathematical foundations, e.g. a detailed error analysis that exists for most of the classical numerical schemes.

Consequently, in this direction, current research has focused on the hybridization of methods, i.e. the combination of traditional numerical methods and DNNs-based approaches in order to further enhance the classical schemes with respect to their efficiency. Let us briefly review some recent developments in the field of numerical solution of linear and nonlinear PDEs using machine learning techniques.

Sirignano and Spiliopoulos [8] proposed a combination of Galerkin methods and DNNs, which they call "Deep Galerkin Method (DGM)", to solve high-dimensional PDEs. The DGM algorithm is meshfree to cope with the curse of dimensionality and is somewhat similar to Galerkin methods, with the solution approximated by a neural network instead of a linear combination of basis functions. In this direction, E and Yu [9] presented the "Deep Ritz Method (DRM)" for numerically solving variational problems in high dimensions. Also, He, Li, Xu and Zheng [10] theoretically analyzed the relationship between DNN with rectified linear unit (ReLU) function as the activation function and the finite element method (FEM). For the proper treatment of the boundary conditions, see [11].

In 2019, Raissi, Perdikaris, and Karniadakis [3] introduced "physics-informed neural networks (PINNs)", a deep-learning framework for synergistically combining mathematical models and data that has found a variety of applications to date. PINNs compute approximate solutions to PDEs by training a neural network to minimize a loss function consisting of terms representing the mismatch of initial and boundary conditions and the PDE residual at chosen points in the interior domain. Later in 2021 Ramabathiran and Ramachandran [12] proposed the "sparse, physics-based, and partially interpretable neural network (SPINN)" model for solving PDEs, which is a new class of hybrid algorithms between PINNs and traditional mesh-free numerical methods. The authors also proposed a hybrid finite difference and SPINN method called FD-SPINN, where the (explicit or implicit) temporal discretization is done using conventional finite difference methods and the spatial discretization is implemented at each time step using the SPINN approach, i.e. the spatial derivatives are handled exactly by automatic differentiation [13].

Long, Lu and Dong [14] proposed PDE-Net to predict the dynamics of complex systems. The underlying PDEs can be discovered from the observational data by making the connections between convolution kernels in convolutional neural networks (CNNs) and differential operators. Based on the integral form of the underlying dynamical system, Qin, Wu, and Xiu [15] considered the ResNet block as a single-stage method and the recurrent ResNet and recursive ResNet as multi-stage methods. Wu and Xiu [16] approximated the evolution operator by a ResNet to solve and recover unknown time-dependent PDEs.

Wang, Shen, Long and Dong [17] used reinforcement learning to empower Weighted Essentially Non-Oscillatory (WENO) schemes to solve 1D scalar conservation laws. In the works [18, 19, 20], the authors have presented a machine learning based approach to further improve the WENO method leading to better approximations of numerical solutions with shocks.

This motivates us to propose new finite difference methods (FDMs) in combination with DNNs. We refer to the new method as the deep finite difference methods (DFDMs). Like some other DNN models, DFDM also learns its representation through supervised pre-training. After the neural network is satisfactorily trained, it is post-processed to predict the solution of the PDE. Let us emphasize that we explicitly capture information about the local truncation error of the FDM instead of directly approaching the solution of the PDE. To the best of our knowledge, this is the first work in which Deep Learning is used to approximate the discretization error in solving PDEs.

In [21] Shen, Cheng and Liang propose a Deep Learning-based algorithm for solving ordinary differential equations (ODEs) based on an approximation of the local truncation error of the Euler scheme, see also [22, 23] for related hypersolver approaches. The basic idea of this method is to augment an ODE solver with a neural network in order to achieve higher accuracy with respect to the time discretization.

While the approximation of the local truncation error is also the core of our method, our approach has several significant differences to [21], which we briefly summarize in the sequel. First, unlike [21], we use the idea of approximating the local truncation error for solving PDEs rather than ODEs. Moreover, we use a different neural network structure, namely a very small CNN, to ensure time efficiency. In [21], a multi-layer fully connected neural network with 8 layers and 80 neurons is used. In our approach, the neural network is trained for a class of PDE problems. The trained method is then applicable to a range of different initial conditions and PDE parameterizations. In [21], the neural network is trained only for a particular ODE problem with a fixed initial condition and for different discretizations. We show that our method generalizes well to different discretizations without the need for retraining. Finally, in [21], the input to the neural network is formed by solving the standard Euler method from the previous time step and using the points that define the time discretization. While we also use the solution from the previous time step as input, we always compute it during the training step, taking into account the influence of the neural network itself. By using CNN, the spatial neighborhood from the previous time step is also part of the input.

The main advantages of the proposed scheme are that the scheme remains convergent and consistent. Although we improve the standard finite difference method (FDM) and compact finite difference method (CFDM), this approach can be easily extended to any traditional numerical scheme. The method is straightforward and very easy to implement. Finally, as a proof of concept, we present some examples and show that the method remains time efficient in most cases despite the addition of the rather small neural network.

The paper is organized as follows. In Section 2, we present the standard FDM approach in detail and explain our deep learning approach that improves the FDM. In Section 3, we introduce the compact FDM and apply our deep learning algorithm to it. In Section 4, we explain the training procedure. Then, in Section 5, we present our numerical results, which are illustrated with tables and figures. Finally, we conclude our work in Section 6.

2. Finite Difference Schemes

Let us consider a (parabolic) PDE of the form

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \sum_{i,j=1}^{d} \alpha_{ij}(\mathbf{x})\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{d} \beta_i(\mathbf{x})\frac{\partial u}{\partial x_i} + \gamma(\mathbf{x})u, \quad (\mathbf{x},t) \in \Omega_d \times [0,T], \\
&u(\mathbf{x},0) = u_0(\mathbf{x}),
\end{aligned} \tag{1}$$

with the coefficients $\alpha_{ij}, \beta_i, \gamma \colon \Omega_d \subseteq \mathbb{R}^d \to \mathbb{R}$, $i,j = 1,\dots,d$, where $\mathbf{x} = (x_1,\dots,x_d) \in \Omega_d$ and d denotes the space dimension. We start with the simple one-dimensional case where the PDE (1) reduces to

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \alpha(x)\frac{\partial^2 u}{\partial x^2} + \beta(x)\frac{\partial u}{\partial x} + \gamma(x)u, \qquad (x,t) \in \Omega_1 \times [0,T], \\
&u(x,0) = u_0(x).
\end{aligned} \tag{2}$$

We select the 1D spatial domain $\Omega_1 = [a,b]$ and introduce a uniform grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0,1,\dots,I$. The time domain $[0,T]$ is discretized uniformly by the points $t_n = t_0 + n\Delta t$, $n = 0,1,\dots,N$. Let us emphasize that uniform grids are considered for simplicity only, our approach can also be applied to nonuniform grids. Let $u_i^n = u(x_i,t_n)$ be the value of the exact solution at the grid point (x_i,t_n) and $\hat{u}_i^n$ be the corresponding numerical approximation.

The simplest numerical approximation of u_i^n can be performed by the *finite difference method*. The well-known second order central approximation to the second derivative is given by

$$\frac{\partial^2 u}{\partial x^2}\Big|_{x_i} = \frac{u(x_{i+1},t) - 2u(x_i,t) + u(x_{i-1},t)}{\Delta x^2} - \frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4}\Big|_{x_i} + O(\Delta x^3), \tag{3}$$

for $u \in C^4(\Omega_1)$ and the central approximation to the first derivative reads

$$\frac{\partial u}{\partial x}\Big|_{x_i} = \frac{u(x_{i+1},t) - u(x_{i-1},t)}{2\Delta x} - \frac{\Delta x^2}{6}\frac{\partial^3 u}{\partial x^3}\Big|_{x_i} + O(\Delta x^3), \tag{4}$$

for $u \in C^3(\Omega_1)$. It can be seen, that the local discretization error $\epsilon_2 = O(\Delta x^2)$ and $\epsilon_1 = O(\Delta x^2)$ is of the second order for both schemes (3) and (4), respectively.

In our work, we propose a deep learning algorithm to improve the accuracy of the above finite difference approximations. To this end, we introduce a neural network trained to approximate the local discretization error ϵ_1 and ϵ_2 such that the final numerical approximation $\hat{u}_i^n$ is improved. Let us abbreviate our resulting new deep learning finite difference method as DFDM. The further details of this method will be discussed in the next section.

Deep Learning used to approximate the FDM discretization error

To ensure the spatial invariance of the proposed scheme and because of its computational efficiency, we use the *convolutional neural network* (CNN). Let $F(\cdot), G(\cdot)\colon \mathbb{R}^{2k+1} \to \mathbb{R}$ be the functions of the CNN, where $2k+1$ is the size of the receptive field (RF) of the CNN. The RF represents the region of the input that affects a particular single element of an output of the CNN [24].

For the temporal discretization, we consider for simplicity the forward Euler scheme, but any other method for solving ODEs could also be used. Now, we discretize the PDE (2) using (3), (4) and adding the neural network function terms $F(\bar{u}_i^n), G(\bar{u}_i^n)$. This leads to the following deep FDM ansatz

$$\begin{aligned} \hat{u}_i^{n+1} = \hat{u}_i^n + \Delta t\Big[\alpha(x_i)\Big(\frac{\hat{u}_{i+1}^n - 2\hat{u}_i^n + \hat{u}_{i-1}^n}{\Delta x^2} + \Delta x^2 F(\bar{u}_i^n)\Big) \\ + \beta(x_i)\Big(\frac{\hat{u}_{i+1}^n - \hat{u}_{i-1}^n}{2\Delta x} + \Delta x^2 G(\bar{u}_i^n)\Big) + \gamma(x_i)\,\hat{u}_i^n\Big], \end{aligned} \tag{5}$$

where $\bar{u}_i^n = \bar{u}^n(\bar{x}_i) = (\hat{u}^n(x_{i-k}), \ldots, \hat{u}^n(x_{i+k})) = (\hat{u}_{i-k}^n, \ldots, \hat{u}_{i+k}^n)$ is the input to the neural network. When applying a CNN kernel to compute $F(\bar{u}_i^n)$ and $G(\bar{u}_i^n)$, under the RF we understand the local neighborhood of $\hat{u}^n(x_i)$

representing input for this computation. For example, if the kernel size of the input CNN layer is 3, the RF of the output of that layer is 3 and $k = 1$ in this case.

Let us note that the functions $F(\bar{u}_i^n)$ and $G(\bar{u}_i^n)$ can share some layers or be represented by the same CNN with two outputs. We train the CNN to fulfill the following approximations:

$$F(\bar{u}_i^n) \approx \frac{1}{\Delta x^2}\,\epsilon_2, \qquad G(\bar{u}_i^n) \approx \frac{1}{\Delta x^2}\,\epsilon_1.$$

and

$$F(\bar{u}_i^n) = G(\bar{u}_i^n) = O(1).$$

The convergence and consistency properties of the standard FDM are preserved. This is ensured due to multiplication of the neural network functions with the step size Δx^2 as in (5). Moreover, the values of the neural network functions have to be bounded, which we will ensure using bounded activation function (such as tanh) in the last CNN layer.

The lowest order terms of discretization errors of (3), (4) can be eliminated by using appropriate difference quotients for these error terms without enlarging the underlying stencil of the scheme. The resulting FDMs of this approach are called 'compact' and will be the topic of the next section.

3. Compact Finite Difference Schemes

Let us consider as benchmark a heat equation of the form

$$\begin{aligned} &\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \qquad (x,t) \in \Omega_1 \times [0,T], \\ &u(x,0) = u_0(x), \end{aligned} \tag{6}$$

with $\alpha > 0$. We select again the spatial domain $\Omega_1 = [a, b]$ with a uniform grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0, 1, \ldots, I$. The time domain $[0, T]$ is discretized uniformly by the points $t_n = t_0 + n\Delta t$, $n = 0, 1, \ldots, N$. To approximate the solution u_i^n we consider now *compact finite difference methods* (CFDMs). The basic idea of these schemes is to further improve the accuracy of traditional FDMs by approximating the lowest order error term by an appropriate difference quotient, without enlarging the stencil dimensions, cf. [25]. For example, the second derivative can be implicitly computed using the fourth-order compact scheme

$$\frac{1}{10}u''_{i+1} + u''_i + \frac{1}{10}u''_{i-1} = \frac{1}{\Delta x^2}\left(\frac{6}{5}u_{i+1} - \frac{12}{5}u_i + \frac{6}{5}u_{i-1}\right) + O(\Delta x^4). \tag{7}$$

Here, the discretization error fulfills $\epsilon = O(\Delta x^4)$ for $u \in C^6(\Omega_1)$.

Deep Learning used to approximate the CFDM discretization error

We describe in this section how our proposed algorithm can be easily generalized to any other standard numerical scheme. We again consider the CNN and add properly the neural network function term to the discretization of the PDE (6). Here, we use for the time discretization the trapezoidal rule, which is second order in time:

$$\frac{\hat{u}^{n+1} - \hat{u}^n}{\Delta t} = \frac{1}{2}\alpha\left(\hat{u}''^{n+1} + \hat{u}''^{n}\right) + \Delta x^4 F(\hat{u}^n), \tag{8}$$

where $F(\hat{u}^n)$ is a vector with elements $F(\hat{u}^n)_i = F(\bar{u}^n_i)$ with $\bar{u}^n_i = \bar{u}^n(\bar{x}_i) = (\hat{u}^n(x_{i-k}), \dots, \hat{u}^n(x_{i+k})) = (\hat{u}^n_{i-k}, \dots, \hat{u}^n_{i+k})$ being the input to the CNN with the size of a receptive field $2k+1$. The factor Δx^4 will be explained at the end of this section. Then, using the discretization scheme (7) and defining the matrices A, B as

$$A = \begin{bmatrix} 1 & \frac{1}{10} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \frac{1}{10} & 1 & \frac{1}{10} & \ddots & & & & \vdots \\ 0 & \frac{1}{10} & 1 & \frac{1}{10} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \frac{1}{10} & 1 & \frac{1}{10} & 0 \\ \vdots & & & & \ddots & \frac{1}{10} & 1 & \frac{1}{10} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & \frac{1}{10} & 1 \end{bmatrix}, \qquad B = \frac{1}{\Delta x^2}\begin{bmatrix} -\frac{12}{5} & \frac{6}{5} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \frac{6}{5} & -\frac{12}{5} & \frac{6}{5} & \ddots & & & & \vdots \\ 0 & \frac{6}{5} & -\frac{12}{5} & \frac{6}{5} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \frac{6}{5} & -\frac{12}{5} & \frac{6}{5} & 0 \\ \vdots & & & & \ddots & \frac{6}{5} & -\frac{12}{5} & \frac{6}{5} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & \frac{6}{5} & -\frac{12}{5} \end{bmatrix}$$

we obtain

$$2\hat{u}^{n+1} - \alpha\Delta t A^{-1}(B\hat{u}^{n+1} + c) = 2\hat{u}^n + \alpha\Delta t A^{-1}(B\hat{u}^n + d) + 2\Delta x^4 \Delta t F(\hat{u}^n), \tag{9}$$

where the vectors c and d represent the boundary conditions for the time steps $n+1$ and n respectively. Using basic matrix operations we obtain

$$(2A - \alpha\Delta t B)\hat{u}^{n+1} = (2A + \alpha\Delta t B)\hat{u}^n + \alpha\Delta t(c+d) + 2A\Delta x^4 \Delta t F(\hat{u}^n). \tag{10}$$

In this case the neural network function is trained to approximate the discretization error of the method such that it holds

$$F(\bar{u}_i^n) \approx \frac{1}{\Delta x^4}\,\epsilon \qquad \text{and} \qquad F(\bar{u}_i^n) = O(1).$$

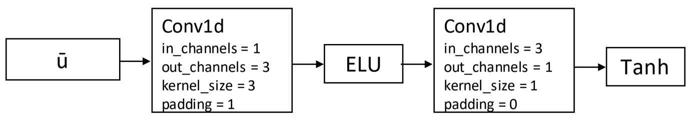
Again, the multiplication of the neural network function $F(\cdot)$ with Δx^4 ensures the fourth order of the enhanced compact scheme, assuming that the neural network output is bounded. This will be again ensured using bounded activation functions in the last CNN layer. Accordingly, we abbreviate our deep learning compact finite difference method as DCFDM.

4. Training procedure

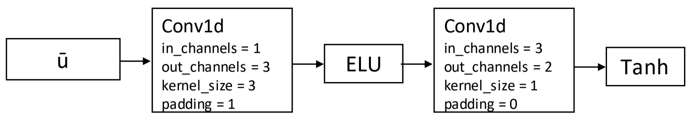
In this section, we describe how the training of the CNN is performed. In our experiments, we use the CNN with only two layers, the input layer and the output layer. The kernel size and the number of channels can be found in Figure 1. This small neural network with a small number of channels ensures numerical efficiency of the resulting hybrid scheme. In a case where the equation contains both a diffusion and a convection term, we use the same neural network to compute the functions $F(\bar{u}_i)$ and $G(\bar{u}_i)$ from (5). These are then represented as two output channels of the neural network, where the first output channel represents the correction of a diffusion term and the second output channel represents the correction of a convection term. In the two-dimensional example, a two-dimensional CNN is used.

At the beginning of the training procedure, the weights of the CNN are initialized randomly. Then, a problem is randomly selected from the dataset. The discrete computational domain is divided into $I \times N$ steps ($I \times J \times N$ for two-dimensional problems), where I, J are the number of space steps in x, y direction and N is the number of time steps. We compute the solution up to a fixed final time T. After each time step n, we predict the discretization error, compute the loss and its gradient with respect to the weights of the CNN, update the weights, and proceed to the next time level $n+1$. At this new time step, a new updated solution according to (5) is the input to the CNN. For the optimization of the loss function we use the Adam optimizer [26], where the learning rate is set separately for each PDE. For the training procedure, we use the mean squared error loss function, defined as

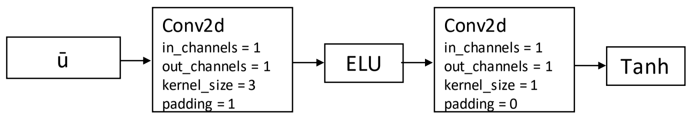
$$\mathrm{LOSS}_{\mathrm{MSE}}(u) = \frac{1}{I}\sum_{i=0}^{I}(\hat{u}_i^n - u_i^{n,\mathrm{ref}})^2, \tag{11}$$



(a) One-dimensional case, equation without convection term.



(b) One-dimensional case, equation with convection term.



(c) Two-dimensional case, equation without convection term.

Figure 1: Structure of the convolutional neural network for different examples.

where $\hat{u}_i^n$ is a numerical approximation of $u(x_i, t_n)$ obtained using DFDM, resp. DCFDM and $u_i^{n,\text{ref}}$ denotes the corresponding reference solution. If the exact solution is available, this is used as the reference solution. Otherwise, the reference solution is calculated on a very fine grid. For the implementation we use Python with the library PyTorch [27]. We summarize the training procedure and the implementation of DFDM in Algorithm 1. The training procedure results in a new numerical scheme, which is then generally applicable for a wide class of PDEs.

Algorithm 1 DFDM training procedure

```
for $l \leftarrow 0$ to $L$ do                    ▷ L: the total number of training cycles
    ◇ choose a new problem from a data set with randomly generated initial condition parameters and/or randomly generated PDE coefficients
    ◇ use fixed final time $T$, spatial and temporal discretization
    for $n \leftarrow 0$ to $N$ do                ▷ N: the total number of time steps
        ◇ Input: Solution $\hat{u}^n$ at time $t_n$
        ◇ evaluation of CNN: Output: discretization error approximation $F(\hat{u}^n)$ or approximations $F(\hat{u}^n), G(\hat{u}^n)$
        ◇ use the equations (5), resp. (10) and compute the solution approximation $\hat{u}^{n+1}$ at time $t_{n+1}$
        ◇ compute loss using equation (11)
        ◇ compute loss gradient with respects to the weights of CNN
        ◇ update weights using chosen optimizer
    end for
    ◇ testing on validation problems
end for
```

5. Numerical Examples

In this section we present our results on a few numerical examples. We provide a detailed comparison of our method with the standard FDM on tables and figures. In all provided tables, we denote as "ratio" the error of the FDM divided by the error of DFDM (rounded to 2 decimal points).

5.1. One-dimensional heat equation

As an introductory example we use the one-dimensional heat equation

$$u_t = u_{xx}, \quad u(x,0) = c + a\sin(b\pi x), \quad -\pi \le x \le \pi, \quad 0 \le t \le T. \tag{12}$$

The exact solution for this example is

$$u(x,t) = c + a\, e^{(-b^2\pi^2 t)} \sin(b\pi x) \tag{13}$$

and we take the boundary conditions from the exact solution for this case.

We proceed during the training as described in Section 4 and specify the learning rate for the Adam optimizer as lr = 0.00001. In a point, where a new problem from a data set should be chosen, we generate the parameters a, b and c randomly such that

$$a \in \mathcal{U}[1, 2], \quad b \in \mathcal{U}[0.3, 0.5], \quad c \in \mathcal{U}[0, 0.25]. \tag{14}$$

We fix the final time $T = 0.25$ for each training cycle. As training cycle we denote a sequence of training steps performed on a solution for an unique problem with randomly chosen parameters a, b and c until the final time T. Then we test the trained model on a validation set, which contains the problems with the parameters not included in the training set. During the training we fix the spatial discretization and divide the spatial domain into $I = 100$ steps. For the temporal discretization we use the relation $\Delta t = 0.5\Delta x^2$, i.e. the parabolic mesh ration $\lambda = \Delta t/\Delta x^2$ is set to 0.5.

We show the evolution of the loss function on the validation set in Figure 2. We run the training for 800 training cycles. Experimentally we found out that the additional training would not improve results anymore. We performed 10 independent trainings and present the results of the training showing the best performance on the validation set. However, let us note, that all trainings have led to a very similar loss evolution. We rescale the loss values for each validation problem to be in the interval $[0, 1]$ using the relation

$$\mathrm{LOSS}_{\mathrm{adjusted}} = \frac{\mathrm{LOSS}^{\mathrm{l}}_{\mathrm{MSE}}(\mathrm{u})}{\max_j(\mathrm{LOSS}^{\mathrm{l}}_{\mathrm{MSE}}(\mathrm{u}))}, \qquad l = 0, \ldots, L, \tag{15}$$

where L denotes the total number of training cycles.

We see that for some initial-value problems the method performs significantly better than for another ones. We choose our model based on validation set of problem. For each of these problems we compute after each training cycle a standard FDM solution and the improvement ratio, defined as the error of the FDM divided by the error of the DFDM. Finally, we choose the model from the training cycle in which the 30% quantile across the improvement ratios of validation problems reaches its maximum. In our case, we took

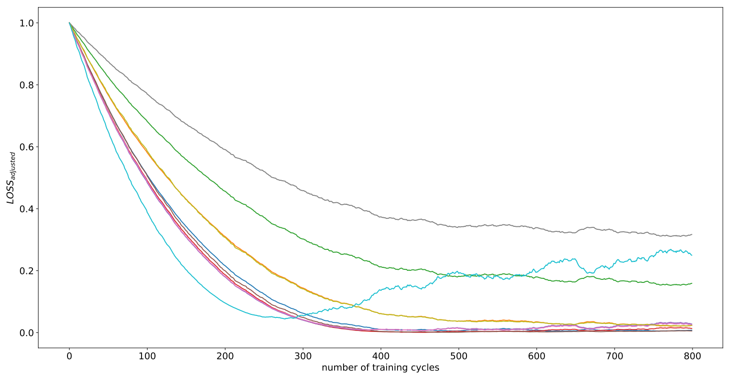


Figure 2: The values of (15) for different validation problems at different training cycles for Example 5.1.

a model obtained after the 685. training cycle and by getting rid of a few problems with a poor improvement we ensure that 70 % validation problems have the improvement ratio 3.27 or bigger. Let us note, that in all presented examples, the same decision rule based on 30 % quantile will be used.

We present the numerical results on problems from the test set for various final times T. These were neither in the training set, nor in the validation set. The Figure 3 illustrates the solution for two different initial value parameters choices. We see, that the method performs well also on the set of parameters a, b, c outside of the training interval. In Table 1 we can see the significant improvement on the errors.

Furthermore, we analyze the computational cost of our method and compare it in Figures 4. We see, that on 7 of 10 examples the DFDM outperforms the standard method. Let us emphasize, that we did not retrain the method for different spatial discretizations.

Next we retrain the neural network for the following diffusion-convection equation

$$u_t = \alpha u_{xx} - \beta u_x, \quad u(x,0) = c + a\sin(b\pi x), \quad 0 \le x \le 2\pi, \quad 0 \le t \le T, \tag{16}$$

where in addition to parameters from (14) also the parameters $\alpha \in \mathcal{U}[1,2]$ and $\beta \in \mathcal{U}[1,2]$ are chosen randomly during the training and testing. The training is performed as described before and we choose the learning rate $\mathrm{lr} = 0.0001$. The CNN structure can be found in Figure 1b and we run the training for 4000 training cycles. We present in Table 2 the results for

parameters			L_∞			L_2		
a	b	c	FDM	DFDM	ratio	FDM	DFDM	ratio
1.88	0.32	0.12	0.000245	0.000353	0.69	0.000433	0.000577	0.75
1.31	0.4	0.12	0.000362	0.000033	11.01	0.000579	0.000049	11.73
1.15	0.43	0.15	0.000398	0.000075	5.31	0.000616	0.000104	5.92
1.95	0.42	0.16	0.000628	0.000179	3.51	0.000981	0.000208	4.71
1.74	0.38	0.02	0.000406	0.000090	4.52	0.000669	0.000151	4.44
1.32	0.41	0.17	0.000394	0.000034	11.74	0.000623	0.000035	17.73
1.43	0.35	0.21	0.000254	0.000239	1.06	0.000435	0.000357	1.22
1.83	0.48	0.078	0.000880	0.000467	1.88	0.001330	0.000688	1.93
1.56	0.39	0.14	0.000396	0.000073	5.47	0.000644	0.000096	6.71
1.53	0.43	0.018	0.000530	0.000151	3.50	0.000820	0.000216	3.79

(a) $T = 0.25$

parameters			L_∞			L_2		
a	b	c	FDM	DFDM	ratio	FDM	DFDM	ratio
1.88	0.32	0.12	0.000380	0.000577	0.66	0.000673	0.000956	0.70
1.31	0.4	0.12	0.000496	0.000072	6.89	0.000817	0.000107	7.65
1.15	0.43	0.15	0.000515	0.000082	6.28	0.000816	0.000121	6.76
1.95	0.42	0.16	0.000828	0.000175	4.74	0.001330	0.000219	6.07
1.74	0.38	0.02	0.000578	0.000190	3.04	0.000975	0.000294	3.32
1.32	0.41	0.17	0.000530	0.000025	21.15	0.000863	0.000034	25.05
1.43	0.35	0.21	0.000379	0.000385	0.98	0.000658	0.000579	1.14
1.83	0.48	0.078	0.001015	0.000535	1.90	0.001518	0.000751	2.02
1.56	0.39	0.14	0.000555	0.000114	4.86	0.000925	0.000173	5.36
1.53	0.43	0.018	0.000685	0.000230	2.98	0.001085	0.000316	3.44

(b) $T = 0.5$

Table 1: Comparison of L_∞ and L_2 error of FDM and DFDM methods for the solution of the heat equation with various parameters a, b, c and T, $I = 100$.

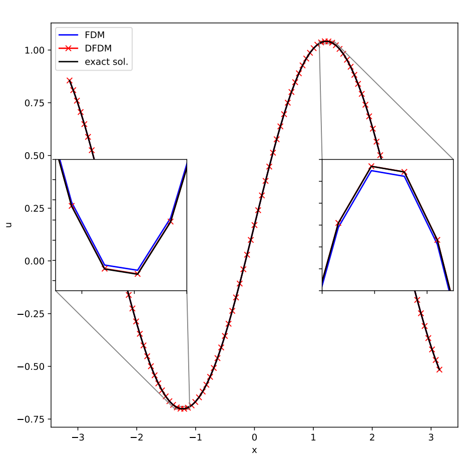
different parametrizations of the PDE and the initial condition (16).

5.2. European Call Option

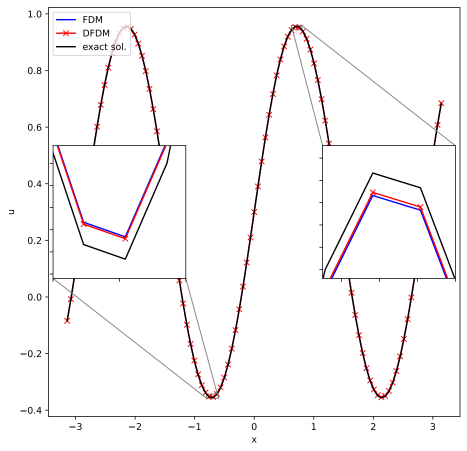
We apply our method also to a problem from computational finance, namely to the option pricing problem. Let us consider the Black-Scholes equation

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV = 0, \quad S \in [0, \infty),\ t \in [0, T], \tag{17}$$

where S is the price of an underlying asset at time t, $r > 0$ is the riskless interest rate and σ^2 is the volatility. In this paper, we solve the European call



(a) Initial condition with $a = 1.32$, $b = 0.41$, $c = 0.17$. (Parameters in the training interval.)



(b) Initial condition with $a = 2.2$, $b = 0.7$, $c = 0.3$. (Parameters outside of the training interval.)

Figure 3: Comparison of the FDM and DFDM methods for the solution of the heat equation, $I = 100$, $T = 0.25$.

option pricing problem with the following terminal and boundary conditions:

$$V(S,T) = \max\{0, S-K\} =: (S-K)^+,$$
$$V(S,t) \to 0, \quad \text{for} \quad S \to 0, \qquad V(S,t) \to S - Ke^{-r(T-t)}, \quad \text{for} \quad S \to \infty, \tag{18}$$

with K being the strike price. We use the following transformation of variables that exploits the Euler structure of the spatial operator in (17) and also reverses the time direction:

$$S = Ke^x, \quad \tau = T - t, \quad V(S,t) = Ku(x,\tau) \tag{19}$$

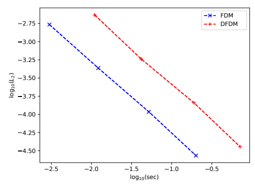
and substitute this into (17) and (18). Then we obtain the (forward-in-time) PDE:

$$u_\tau = \frac{\sigma^2}{2}u_{xx} + \left(r - \frac{\sigma^2}{2}\right)u_x - ru, \quad x \in \mathbb{R},\ 0 \le \tau \le T. \tag{20}$$

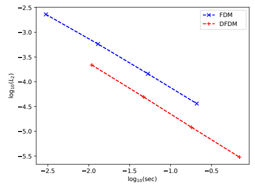
For the training, we generate randomly the parameters

$$\sigma \in \mathcal{U}[0.4, 0.6], \quad r \in \mathcal{U}[0.1, 0.3]. \tag{21}$$

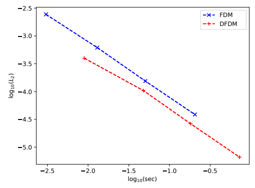
Further, we set $K = 80$, $T = 1$ and divide the computational domain $[x_L, x_R] = [-2, 1.5]$ into 50 space steps and use the temporal step size $\Delta\tau = 0.8\Delta x^2/\sigma^2$.



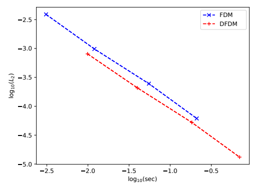
(a) $a = 1.88$, $b = 0.32$, $c = 0.12$



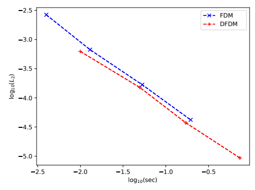
(b) $a = 1.31$, $b = 0.4$, $c = 0.12$



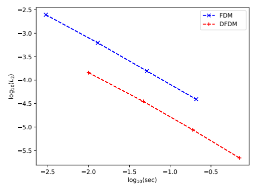
(c) $a = 1.15$, $b = 0.43$, $c = 0.15$



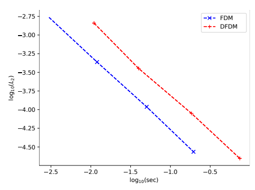
(d) $a = 1.95$, $b = 0.42$, $c = 0.16$



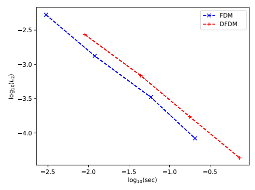
(e) $a = 1.74$, $b = 0.38$, $c = 0.02$



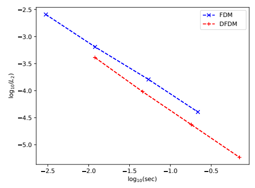
(f) $a = 1.32$, $b = 0.41$, $c = 0.17$



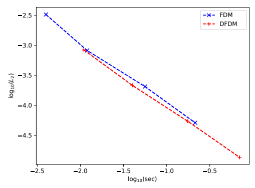
(g) $a = 1.43$, $b = 0.35$, $c = 0.21$



(h) $a = 1.83$, $b = 0.48$, $c = 0.078$



(i) $a = 1.56$, $b = 0.39$, $c = 0.14$



(j) $a = 1.53$, $b = 0.43$, $c = 0.018$

Figure 4: Comparison of computational cost against L_2-error of the solution of the heat equation with various parameters a, b and c according to Table 1. $T = 0.25$.

During training we use the neural network structure as in Figure 1b. We use the learning rate lr $= 0.0001$ and run the training for 4000 training cycles with fixed final time $T = 1$. Figure 5 shows the evolution of the loss function. Using the same decision rule for the best model as in Example 5.1 we choose the model obtained after the 1532. training cycle. Numerical results on problems from the test set can be found in Table 3.

5.3. *Two-dimensional heat equation*

Here we extend the example from the Section 5.1 to two space dimensions. We solve the following two-dimensional heat equation

$$\begin{aligned} u_t &= u_{xx} + u_{yy}, \\ u(x,0) &= c + a\sin(b\pi x) + d\sin(e\pi y), \quad -\pi \le x, y \le \pi, \quad 0 \le t \le T. \end{aligned} \tag{22}$$

parameters					L_∞			L_2		
α	β	a	b	c	FDM	DFDM	ratio	FDM	DFDM	ratio
1.05	1.12	1.08	0.36	0.15	0.000297	0.000123	2.41	0.000465	0.000168	2.76
1.17	1.51	1.12	0.48	0.04	0.000692	0.000200	3.47	0.001043	0.000292	3.57
1.24	1.46	1.51	0.35	0.05	0.000507	0.000160	3.18	0.000774	0.000197	3.92
1.32	1.17	1.69	0.48	0.18	0.000789	0.000324	2.44	0.001243	0.000523	2.38
1.48	1.81	1.78	0.34	0.04	0.000673	0.000203	3.31	0.000991	0.000264	3.75
1.51	1.68	1.21	0.47	0.1	0.000667	0.000188	3.55	0.001033	0.000308	3.35
1.6	1.72	1.75	0.39	0.08	0.000709	0.000047	15.01	0.001112	0.000061	18.16
1.72	1.24	1.9	0.45	0.16	0.000751	0.000210	3.58	0.001222	0.000342	3.58
1.84	1.36	1.32	0.38	0.21	0.000378	0.000108	3.51	0.000560	0.000166	3.37
1.96	1.91	1.41	0.43	0.17	0.000633	0.000107	5.93	0.001009	0.000159	6.33

Table 2: Comparison of L_∞ and L_2 error of FDM and DFDM methods for the solution of the diffusion-convection equation (16) with various parameters α, β, a, b, c, $I = 100$, $T = 0.25$.

For a training we again generate randomly the following parameters

$$a \in \mathcal{U}[1,2], \quad b \in \mathcal{U}[0.3,0.5], \quad c \in \mathcal{U}[0,0.25] \quad d \in \mathcal{U}[1,2], \quad e \in \mathcal{U}[0.3,0.5]$$

and fix the final time $T = 0.25$ and the uniform spatial discretization $I \times J = 50 \times 50$ for each training cycle. In this case we use *two-dimensional CNN* with the parameters which can be found in Figure 1c. As one can see, we only use very small CNNs with only one input layer and output layer and with only one channel in each layer. We set the learning rate for the Adam optimizer lr $= 0.00005$. Training is performed as described before in the one-dimensional example and we run it for 3000 training cycles. We again choose the model with the best performance on the validation set as described in Example 5.1 and present the numerical results on problems from the test set in Table 4 and in Figure 6.

5.4. One-dimensional heat equation with deep compact finite difference method

In the last example we apply the DCFDM to the 1D heat equation

$$u_t = \alpha u_{xx}, \quad u(x,0) = c + a\sin(b\pi x), \quad -\pi \le x \le \pi, \quad 0 \le t \le T. \tag{23}$$

During training and testing the parameters α, a, b and c are chosen randomly such that

$$\alpha \in \mathcal{U}[1,2], \quad a \in \mathcal{U}[1,2], \quad b \in \mathcal{U}[0.3,0.5], \quad c \in \mathcal{U}[0,0.25]. \tag{24}$$

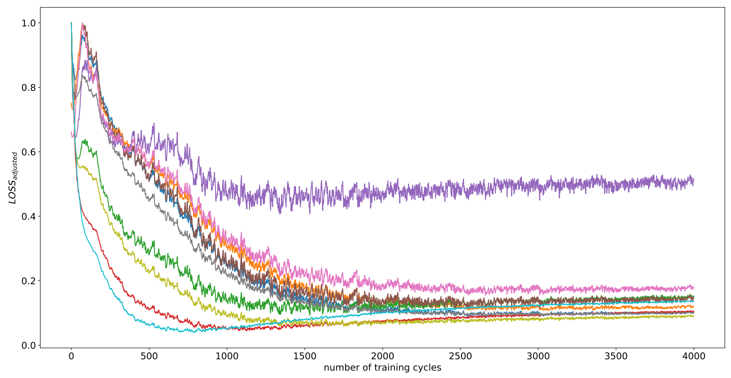


Figure 5: The values of (15) for different validation problems at different training cycles for Example 5.2.

parameters		L_∞			L_2		
σ	r	FDM	DFDM	ratio	FDM	DFDM	ratio
0.48	0.17	0.000682	0.000138	4.95	0.000655	0.000128	5.13
0.59	0.21	0.000629	0.000590	1.07	0.000664	0.000291	2.28
0.49	0.19	0.000707	0.000157	4.50	0.000705	0.000141	5.00
0.55	0.18	0.000611	0.000324	1.89	0.000607	0.000179	3.39
0.41	0.10	0.000632	0.000280	2.26	0.000503	0.000202	2.49
0.43	0.26	0.000977	0.000318	3.08	0.001089	0.000260	4.20
0.45	0.15	0.000680	0.000136	4.99	0.000622	0.000136	4.56
0.52	0.22	0.000740	0.000200	3.70	0.000767	0.000185	4.14
0.54	0.14	0.000544	0.000346	1.57	0.000510	0.000165	3.10
0.46	0.24	0.000880	0.000251	3.51	0.000944	0.000207	4.55

Table 3: Comparison of L_∞ and L_2 error of FDM and DFDM methods for the solution of the Black-Scholes equation (16) with various parameters σ and r, $I = 50$, $T = 1$.

Further, for the training we set $T = 0.25$, divide the computational domain into 25 space steps and for the temporal step size use $\Delta t = \Delta x^2$, i.e. $\lambda = 1$. The neural network structure is used as in the Figure 1a. We select the learning rate lr $= 0.0001$ and run the training for 2000 training cycles. We choose the final model according to the rule in the Example 5.1 and present the results in Table 5.

We see, that the neural network can improve the CFDM very well. Due to the implicitness of the method, the time complexity which is added through CNN is not that big compared to the time complexity of the classical finite difference method. As illustrated in Figure 7, the DCFDM remains time effective in most of the cases. We note, that we did not retrain the method

parameters					L_∞			L_2		
a	b	c	d	e	FDM	DFDM	ratio	FDM	DFDM	ratio
1	0.41	0	1.2	0.4	0.000611	0.000159	3.85	0.000250	0.000059	4.26
1.7	0.42	0.05	1.2	0.45	0.000994	0.000430	2.31	0.000395	0.000166	2.38
1.02	0.35	0	1.51	0.4	0.000580	0.000148	3.93	0.000259	0.000060	4.34
1.98	0.45	0.1	1.02	0.38	0.000996	0.000408	2.44	0.000444	0.000217	2.05
1.63	0.4	0.1	1.1	0.5	0.001014	0.000493	2.06	0.000407	0.000221	1.85
1.42	0.37	0.06	1.01	0.43	0.000634	0.000169	3.74	0.000261	0.000077	3.39
1.52	0.36	0.15	1.6	0.48	0.001034	0.000553	1.87	0.000446	0.000256	1.75
1.12	0.4	0.24	1.83	0.31	0.000505	0.000388	1.30	0.000220	0.000174	1.26
1.21	0.32	0.18	1.8	0.38	0.000559	0.000194	2.87	0.000263	0.000095	2.76
1.91	0.44	0.03	1.79	0.44	0.001337	0.000655	2.04	0.000525	0.000244	2.15

(a) $T = 0.25$

parameters					L_∞			L_2		
a	b	c	d	e	FDM	DFDM	ratio	FDM	DFDM	ratio
1	0.41	0	1.2	0.4	0.000818	0.000209	3.92	0.000333	0.000078	4.30
1.7	0.42	0.05	1.2	0.45	0.001258	0.000525	2.40	0.000495	0.000203	2.44
1.02	0.35	0	1.51	0.4	0.000800	0.000213	3.76	0.000353	0.000083	4.24
1.98	0.45	0.1	1.02	0.38	0.001256	0.000520	2.41	0.000540	0.000258	2.09
1.63	0.4	0.1	1.1	0.5	0.001217	0.000575	2.12	0.000470	0.000229	2.05
1.42	0.37	0.06	1.01	0.43	0.000850	0.000225	3.77	0.000348	0.000098	3.56
1.52	0.36	0.15	1.6	0.48	0.001259	0.000676	1.86	0.000512	0.000282	1.81
1.12	0.4	0.24	1.83	0.31	0.000717	0.000611	1.17	0.000308	0.000269	1.15
1.21	0.32	0.18	1.8	0.38	0.000796	0.000317	2.51	0.000369	0.000150	2.46
1.91	0.44	0.03	1.79	0.44	0.001670	0.000787	2.12	0.000646	0.000296	2.18

(b) $T = 0.5$

Table 4: Comparison of L_∞ and L_2 error of FDM and DFDM methods for the solution of two-dimensional heat equation with various parameters a, b, c, d, e and T. $I \times J = 50 \times 50$.

for different spatial discretizations.

6. Conclusion

In this work we developed a new deep-learning based finite difference scheme for solving PDEs. This numerical scheme is based on an approximation of the local discretization error and remains consistent and convergent. We have shown that this approach can easily be extended to other numerical methods, e.g. compact FDMs. This scheme is easy to use and provides improved numerical results which are demonstrated on the presented examples. We show that the method is able to generalize well, i.e. it yields good results for parameters outside the training region, and remains time efficient even

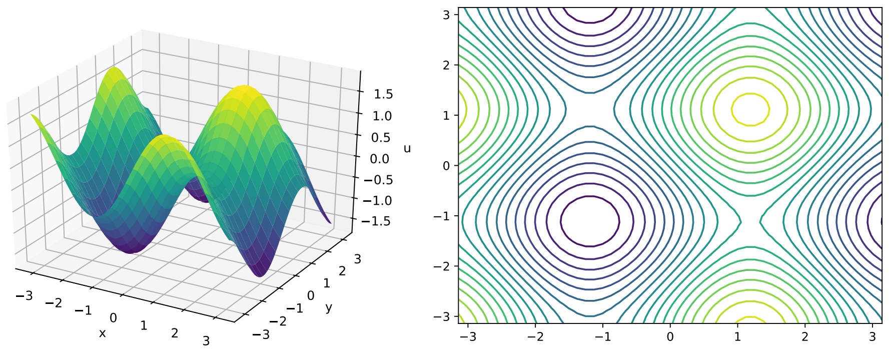


Figure 6: Solution of two-dimensional heat equation (22) with parameters $a = 1.7$, $b = 0.42$, $c = 0.05$, $d = 1.2$, $e = 0.45$. $I \times J = 50 \times 50$, $T = 0.25$.

parameters				L_∞			L_2		
α	a	b	c	CFDM	DCFDM	ratio	CFDM	DCFDM	ratio
1.59	1.88	0.32	0.12	0.000026	0.000021	1.26	0.000046	0.000035	1.30
1.17	1.31	0.4	0.12	0.000026	0.000010	2.56	0.000041	0.000016	2.51
1.71	1.15	0.43	0.15	0.000081	0.000058	1.39	0.000127	0.000088	1.44
1.09	1.95	0.42	0.16	0.000040	0.000004	9.40	0.000063	0.000007	9.03
1.33	1.74	0.38	0.02	0.000037	0.000009	4.28	0.000061	0.000012	4.99
1.41	1.32	0.41	0.17	0.000047	0.000017	2.82	0.000075	0.000024	3.06
1.63	1.43	0.35	0.21	0.000034	0.000006	5.65	0.000058	0.000009	6.55
1.91	1.83	0.48	0.078	0.000257	0.000232	1.11	0.000386	0.000345	1.12
1.80	1.56	0.39	0.14	0.000079	0.000046	1.72	0.000132	0.000076	1.74
1.21	1.53	0.43	0.018	0.000047	0.000014	3.34	0.000072	0.000017	4.28

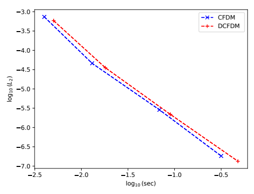
Table 5: Comparison of L_∞ and L_2 error of CFDM and DCFDM methods for the solution of the heat equation with various parameters α, a, b, c, $T = 0.25$, $I = 50$.

though the small neural network part is added.

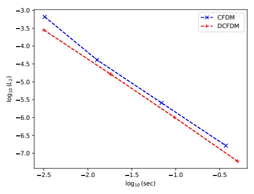
Finally, let us note that this paper can be seen as a proof of concept that the deep learning can be easily used to approximate the local discretization error of the numerical scheme for solving PDEs. In our future work we will further investigate this approach in more detail by applying it to more innovative numerical schemes and facing more challenging examples.

References

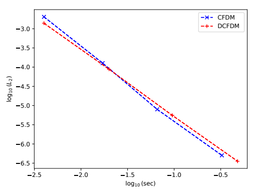
[1] J. Blechschmidt, O. G. Ernst, Three ways to solve partial differential equations with neural networks – a review, GAMM-Mitteilungen 44 (2021) e202100006.



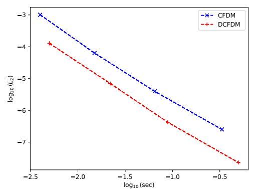
(a) $\alpha = 1.59$, $a = 1.88$, $b = 0.32$, $c = 0.12$



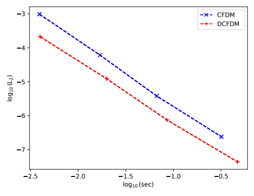
(b) $\alpha = 1.17$, $a = 1.31$, $b = 0.4$, $c = 0.12$



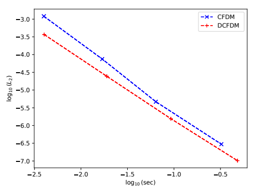
(c) $\alpha = 1.71$, $a = 1.15$, $b = 0.43$, $c = 0.15$



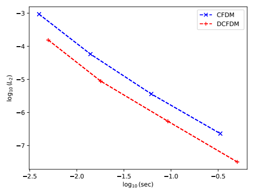
(d) $\alpha = 1.09$, $a = 1.95$, $b = 0.42$, $c = 0.16$



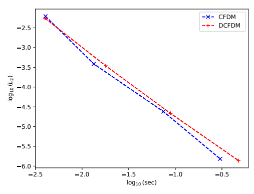
(e) $\alpha = 1.33$, $a = 1.74$, $b = 0.38$, $c = 0.02$



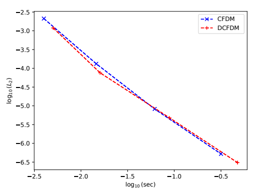
(f) $\alpha = 1.41$, $a = 1.32$, $b = 0.41$, $c = 0.17$



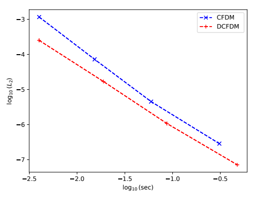
(g) $\alpha = 1.63$, $a = 1.43$, $b = 0.35$, $c = 0.21$



(h) $\alpha = 1.91$, $a = 1.83$, $b = 0.48$, $c = 0.078$



(i) $\alpha = 1.80$, $a = 1.56$, $b = 0.39$, $c = 0.14$



(j) $\alpha = 1.21$, $a = 1.53$, $b = 0.43$, $c = 0.018$

Figure 7: Comparison of computational cost of against L_2-error on the solution of the heat equation with various parameters α, a, b and c according to Table 5, $T = 0.25$.

[2] C. Beck, M. Hutzenthaler, A. Jentzen, B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, Discr. Contin. Dynam. Syst.-B 28 (2023) 3697–3746. doi:`10.3934/dcdsb.2022238`.

[3] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[4] A. B. Farimani, J. Gomes, V. S. Pande, Deep learning the physics of transport phenomena, arXiv preprint arXiv:1709.02432 (2017).

[5] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, Europ. J. Appl. Math. 32 (2021) 421–435.

[6] Y. Sun, L. Zhang, H. Schaeffer, NeuPDE: neural network based ordinary and partial differential equations for modeling time-dependent data, in: Mathematical and Scientific Machine Learning, PMLR, 2020, pp. 352–372.

[7] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (2018) 932–955.

[8] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.

[9] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12. doi:`10.1007/s40304-018-0127-z`.

[10] J. He, L. Li, J. Xu, C. Zheng, ReLU deep neural networks and linear finite elements, arXiv preprint arXiv:1807.03973 (2018).

[11] Y. L. Ming, et al., Deep Nitsche Method: Deep Ritz Method with essential boundary conditions, Commun. Comput. Phys. 29 (2021) 1365–1384. doi:`10.4208/cicp.OA-2020-0219`.

[12] A. A. Ramabathiran, P. Ramachandran, SPINN: sparse, physics-based, and partially interpretable neural networks for PDEs, J. Comput. Phys. 445 (2021) 110600.

[13] A. Griewank, et al., On automatic differentiation, Mathematical Programming: recent developments and applications 6 (1989) 83–107.

[14] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.

[15] T. Qin, K. Wu, D. Xiu, Data driven governing equations approximation using deep neural networks, J. Comput. Phys. 395 (2019) 620–635.

[16] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, J. Comput. Phys. 408 (2020) 109307.

[17] Y. Wang, Z. Shen, Z. Long, B. Dong, Learning to discretize: solving 1D scalar conservation laws via deep reinforcement learning, arXiv preprint arXiv:1905.11079 (2019).

[18] T. Kossaczká, M. Ehrhardt, M. Günther, Enhanced fifth order WENO shock-capturing schemes with deep learning, Res. Appl. Math. 12 (2021) 100201.

[19] T. Kossaczká, M. Ehrhardt, M. Günther, A neural network enhanced weighted essentially non-oscillatory method for nonlinear degenerate parabolic equations, Physics of Fluids 34 (2022) 026604.

[20] T. Kossaczká, M. Ehrhardt, M. Günther, A deep smoothness WENO method with applications in option pricing, in: M. Ehrhardt, M. Günther (Eds.), Progress in Industrial Mathematics at ECMI 2021, Springer, 2022, pp. 417–423.

[21] X. Shen, X. Cheng, K. Liang, Deep Euler method: solving ODEs by approximating the local truncation error of the Euler method, arXiv preprint arXiv:2003.09573 (2020).

[22] F. Zhao, X. Chen, J. Wang, Z. Shi, S.-L. Huang, Performance-guaranteed ODE solvers with complexity-informed neural networks, in: The Symbiosis of Deep Learning and Differential Equations, 2021, pp. 1–6.

[23] J. Kadupitiya, G. C. Fox, V. Jadhao, Solving Newton's equations of motion with large timesteps using recurrent neural networks based operators, Machine Learning: Science and Technology 3 (2022) 025002.

[24] A. Araujo, W. Norris, J. Sim, Computing receptive fields of convolutional neural networks, Distill 4 (2019) e21.

[25] S. K. Lele, Compact finite difference schemes with spectral-like resolution, J. Comput. Phys. 103 (1992) 16–42.

[26] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[27] A. Paszke, et al., PyTorch: An imperative style, high-performance deep learning library, in: H. Wallach, et al. (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.