Deep Learning Enhanced Numerical Schemes

Dissertation

zur Erlangung
des akademischen Grades
eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

der
Fakultät für Mathematik und Naturwissenschaften
der
Bergischen Universität Wuppertal (BUW)
vorgelegt von

Tatiana Kossaczká, M. Sc.
geb. Čechvalová

geboren am 29.12.1993 in Trenčín, Slowakei

Erstbetreuer: Prof. Dr. Matthias Ehrhardt (BUW)
Zweitbetreuer: Prof. Dr. Michael Günther (BUW)

Wuppertal, 2024

Abstract

In this thesis we are concerned with the improvement of existing numerical schemes using deep learning. In the first part we propose a new idea to enhance standard numerical methods for solving partial differential equations (PDEs) with a deep learning approach. The idea is based on an approximation of the local truncation error of the numerical methods used to approximate the spatial derivatives of a given PDE. We present our idea as a proof of concept for improving the standard and compact finite difference methods (FDMs), but it can be easily generalized to other standard numerical methods as well. 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. We also perform a time complexity analysis and show the efficiency of our method.

In the second part we improve the fifth-order Weighted Essentially Non-Oscillatory (WENO) shock capturing scheme for solving hyperbolic conservation laws by integrating deep learning techniques. We improve the established WENO algorithm by training a compact neural network to dynamically adjust the smoothness indicators within the WENO scheme. This modification increases the accuracy of the numerical results, especially in the vicinity of abrupt shocks. In particular, our approach eliminates the need for additional post-processing steps. We call our new method by WENO-DS (Deep Smoothness). We demonstrate the superiority of WENO-DS by examining several examples from the literature on the one- and two-dimensional Buckley-Leverett and Burgers' equations, as well as the Euler equations of gas dynamics. Through a thorough study of these test problems, which include various shocks and rarefaction waves, our novel technique consistently outperforms the traditional fifth-order WENO scheme.

We also extend our approach and apply it to the sixth-order WENO scheme for solving nonlinear degenerate parabolic equations. Our results are presented on benchmark examples of nonlinear degenerate parabolic equations, such as the equation of a porous medium with the Barenblatt solution, the Buckley-Leverett equation and their extensions in two-dimensional space. It is shown that in our experiments the new method outperforms the standard WENO method, reliably handles the sharp interfaces, and provides good resolution of discontinuities.

For both WENO-DS methods we present the corresponding theoretical proofs of fifth- and sixth-order of accuracy. Finally, we demonstrate the applicability and effectiveness of WENO-DS in computational finance problems.

Let us finally note that this thesis is based on the following peer-reviewed publications

- T. Kossaczká, M. Ehrhardt, and M. Günther. Enhanced fifth order WENO shock-capturing schemes with deep learning. *Results in Applied Mathematics*, 12:100217, 2021.
- T. Kossaczká, M. Ehrhardt, and M. Günther. A neural network enhanced WENO method for nonlinear degenerate parabolic equations. *Physics of Fluids*, 34(2):026604, 2022.
- T. Kossaczká, M. Ehrhardt, and M. Günther. A deep smoothness WENO method with applications in option pricing. In *Progress in Industrial Mathematics at ECMI 2021*, pages 417-423. Springer, 2022.
- T. Kossaczká, M. Ehrhardt, and M. Günther. Deep FDM: Enhanced finite difference methods by deep learning. *Franklin Open*, 4:100039, 2023.
- T. Kossaczká, A.D. Jagtap, and M. Ehrhardt. Deep smoothness weighted essentially non-oscillatory method for two-dimensional hyperbolic conservation laws: A deep learning approach for learning smoothness indicators. *Physics of Fluids*, 36(3):036603, 2024.
- T. Kossaczká, M. Ehrhardt, and M. Günther. Deep finite difference method for solving Asian option pricing problems. To appear in *Progress in Industrial Mathematics at ECMI 2023*, Springer, 2024.

Acknowledgements

First of all I would like to thank my supervisor Prof. Dr. Matthias Ehrhardt. I am grateful, that he gave me the opportunity to write my Ph.D. thesis at the Department of Applied and Computational Mathematics (ACM) at Bergische Universität Wuppertal. Moreover, I would like to thank him for already supervising my master's thesis in the research area of Weighted Essentially Non-Oscillatory Method and for encouraging me to continue my doctoral research. Throughout my studies, he accompanied me with beneficial discussions and provided me with valuable feedback.

Next, I would like to thank Prof. Dr. Michael Günther for all his support and for giving me the opportunity to be a part of ACM. I would like to express my thanks to Dr. Ameya D. Jagtap for sharing his knowledge with me, as well as for a valuable collaboration.

I would also like to thank all my colleagues at ACM. Through scientific discussions with them, I always had the opportunity to learn something new, and through nice conversations with them, I could spend a pleasant time at the university.

Furthermore, I would like to thank all the lecturers at Comenius University in Bratislava that I had the pleasure to meet during my bachelor studies. They gave me the essential knowledge that I needed to start my master's studies in Germany, and I appreciate it very much.

I would like to express my deepest thanks to my family, my husband Igor, and my children Terézia and Timotej. I am most grateful to Igor for his great support during my studies and for his unconditional love. Terézia and Timotej were the most wonderful companions with whom I could spend the most beautiful moments, which motivated me throughout my research.

Next, I would like to thank my family in Slovakia, especially my parents for raising me up and supporting me, and my sisters for all the beautiful moments we shared together. I would also like to thank all my other friends that I am glad to have.

Contents

Abstract I

Acknowledgements III

Contents V

1 Introduction 1
- 1.1 Introduction to numerical mathematics 1
- 1.2 Introduction to deep learning 2
  - 1.2.1 Neural network architectures 2
  - 1.2.2 Activation functions 4
  - 1.2.3 Loss function 5
  - 1.2.4 Optimizer and the optimal learning rate 5
  - 1.2.5 Backpropagation algorithm 5
  - 1.2.6 Possible improvements and modifications 6
  - 1.2.7 Combination of numerical methods with deep learning 6
- 1.3 Outline of the thesis 6

I Enhanced finite difference methods by deep learning 9

2 Introduction to deep learning based numerical algorithms and literature overview 11

3 Deep finite difference method 15
- 3.1 Standard finite difference method 15
  - 3.1.1 Deep learning used to approximate the FDM discretization error 16
- 3.2 Compact finite difference method 17
  - 3.2.1 Deep learning used to approximate the CFDM discretization error 18
- 3.3 Training procedure 19
- 3.4 Numerical examples 21
  - 3.4.1 One-dimensional heat equation 21
  - 3.4.2 European call option 25
  - 3.4.3 Two-dimensional heat equation 26
  - 3.4.4 One-dimensional heat equation with Deep CFDM 28
- 3.5 Asian option pricing problem and modification of Deep FDM 28
  - 3.5.1 Modification of the scheme and training procedure 30

3.5.2 Numerical results 31

II Enhanced weighted essentially non-oscillatory schemes by deep learning 35

4 Introduction to hyperbolic conservation laws and literature overview 37
4.1 Mathematical background for hyperbolic conservation laws 37
4.2 Numerical methods for hyperbolic conservation laws 39
    4.2.1 Conservative methods 41
4.3 Related scientific work 43

5 Fifth-order WENO scheme for hyperbolic conservation laws 47
5.1 The WENO scheme 47
    5.1.1 Fifth-order WENO scheme 49
    5.1.2 Smoothness indicators 50
    5.1.3 The WENO-Z scheme 52
5.2 Application of deep learning to the fifth-order WENO scheme . . . 53
    5.2.1 Accuracy analysis 54
5.3 Structure of neural network 57
5.4 The training procedure 60
5.5 Two-dimensional implementation 61
5.6 Numerical results 61
    5.6.1 The Buckley-Leverett equation 62
    5.6.2 The inviscid Burgers' equation 69
    5.6.3 The two-dimensional Burgers' equation 72

6 Deep smoothness WENO scheme for Euler system 75
6.1 One-dimensional Euler system 75
    6.1.1 Structure of neural network and the training procedure . . . 77
    6.1.2 Numerical examples 81
6.2 Two-dimensional Euler system 85
    6.2.1 Structure of neural network and the training procedure . . . 86
    6.2.2 Numerical examples 88

7 Sixth-order WENO scheme for nonlinear degenerate parabolic equations 105
7.1 The WENO scheme 106
    7.1.1 Sixth-order WENO scheme 107
    7.1.2 Smoothness indicators 108
    7.1.3 The MWENO scheme 110
7.2 Application of deep learning to the sixth-order WENO scheme . . . 110
    7.2.1 Accuracy analysis 111
7.3 Structure of neural network 113
7.4 The training procedure 113
7.5 Two-dimensional implementation 114

7.6 Numerical results 115
  7.6.1 The porous medium equation with the Barenblatt solution . 116
  7.6.2 Interaction of two boxes for the porous medium equation . . 120
  7.6.3 The convection-diffusion Buckley-Leverett equation 121
  7.6.4 The strongly degenerate parabolic convection-diffusion equation 125
  7.6.5 Two-dimensional porous medium equation with the Barenblatt solution 127
  7.6.6 Interaction of two bumps for two-dimensional porous medium equation 128
  7.6.7 Two-dimensional Buckley-Leverett equation 129

8 Deep smoothness WENO scheme with application in computational finance 133
8.1 Problem formulation 134
8.2 Application of deep learning and training procedure 134
8.3 Numerical results 136

9 Conclusion and outlook 139
9.1 Highlights of the thesis and outlook 140

References 143

1 Chapter 1

Introduction

1.1 Introduction to numerical mathematics

Many practical problems, e.g. in quantitative finance, stochastic control and quantum physics, can be modelled by partial differential equations (PDEs) which in most cases do not admit analytical solutions. It is therefore 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 ratio of the accuracy achieved to the computational time required.

According to the main classification of PDEs, there are elliptic, hyperbolic and parabolic PDEs. For each of these main types there are different solution strategies. It is very important to choose the most appropriate numerical method for a particular type of PDE. The basic idea of many numerical methods is to first approximate the spatial derivatives of the underlying PDE. Then, for time-dependent PDEs, the system of ordinary differential equations (ODEs) is obtained. This system can be solved using, for example, the explicit or implicit Euler scheme or Runge-Kutta method.

One of the most straightforward methods to approximate the spatial derivatives of PDEs is the finite difference method (FDM). The entire spatial domain must be discretized and the value of the solution is then approximated at each of these discrete points. The difference formulas are constructed by Taylor expansion, assuming smooth solutions. Depending on the number of spatial points involved, we can obtain finite difference approximations of different orders. Convergence, consistency and stability analysis have to be investigated.

In this thesis we consider two main classes of classical numerical methods, finite difference and weighted essentially non-oscillatory (WENO) methods, for solving PDEs. Our main focus is to improve the standard numerical schemes so that their original accuracy and stability properties are preserved. We focus on the important parameters of these methods and aim to improve these parameters using deep learning techniques. In the following section, we provide a brief overview of the core concept of deep learning and outline its potential application in improving conventional numerical techniques.

1.2 Introduction to deep learning

In this thesis, we aim to improve standard numerical methods using deep learning. To better understand our approach, we will briefly introduce deep learning in this section. Deep learning is a part of machine learning based on the ability to learn from data. Unlike other machine learning techniques, such as linear regression or decision trees, deep learning uses artificial neural networks (NNs) with multiple layers. In the learning process, the data is fed to the NN and in each layer it is being processed in increasingly abstract ways.

The NN consisting of only one layer can be seen as a classical linear regression. Application of an *activation function* to the linear regression output would form the NN architecture called *Perceptron*. It was invented in 1957 by Frank Rosenblatt [91] and mathematically can be described by:

$$y = \Omega\big(\sum_{i=1}^{n} w_i \times x_i + b\big), \tag{1.1}$$

where y is a NN output, x_i represent inputs, w_i correspond to the weights associated with each input, b is a bias term, n is the number of inputs and $\times$ denotes the element-wise multiplication. The activation function Ω adds the non-linearity to the output. Other NN structures can be understood as a composition and organization of Perceptrons into layers, creating more complicated architectures. NN structure consisting of one input layer, one or more hidden layers and one output layer is called Multi-Layer Perceptron (MLP).

1.2.1 Neural network architectures

The most common NN architecture is *a fully connected neural network*, also known as *a dense neural network*. It is actually a MLP and the terms are often used interchangeably. This NN consists of several layers of neurons, where each neuron in one layer is connected to each neuron in the next layer. The first layer is the input layer, the last layer is the output layer, and all the layers in between are called hidden layers. This NN structure is illustrated in Figure 1.1.

Second widely used NN architecture is *a convolutional neural network* (CNN). They are very commonly used e.g. in image processing or object classification, as they can capture local patterns and features efficiently. CNNs have fewer parameters compared to fully connected networks of equivalent size, thanks to weight sharing across different spatial locations of the input data, making them more computationally efficient and easier to train, especially on large datasets.

Let us now describe the main properties of CNNs. In CNNs, *channels* play a crucial role in processing and representing data. Considering the input data, channels refer to its different features or aspects. For example, in numerical examples, with more

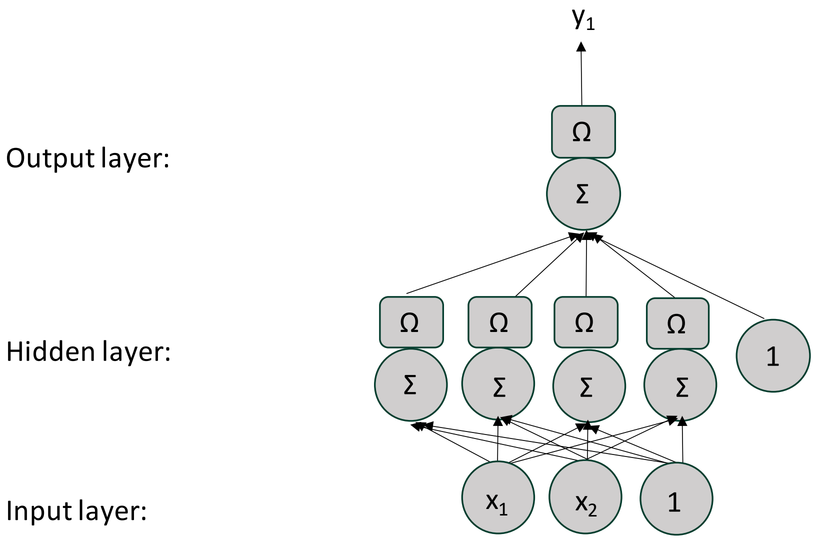


Figure 1.1: Multi-Layer Perceptron.

channels on the input, each could represent a different variable and/or derivative of a particular variable. CNN then allows the extracted information to be combined across different features of the data. Each of the output channels of a last hidden layer represents a different data feature that the network is learning. We can increase the complexity of the CNN by increasing the number of channels in hidden layers, which would capture different combinations of features learned from lower layers.

Next, CNN uses a *kernel*, a small window of parameters that slides across the input data, to extract local patterns or features. The kernel size for each CNN layer then affects the size of *receptive field* of the CNN. The receptive field represents the region of the input that affects a particular single element of an output of the CNN [5]. Further, *stride* and *padding* parameters need to be specified. Stride refers to the step size at which the kernel moves over the input data. Padding refers to the number of artificial data values added to the boundary of the input data. Padding can be constant or data-dependent and essentially plays role of numerical boundary conditions. An schematic, which explains the role of padding, kernel and stride can be found in Figure 1.2. We illustrate a kernel with size 5, which is moved always by 3 input units, so the stride is 3. We used padding 2, so that the output is of size 3. To obtain the output of the same size as the input (7) we would have to use padding of size 4.

Although we will mainly use one-dimensional CNNs, in practice, *two-dimensional* CNNs are also widely used, especially for image processing. In this case, input data is typically represented as a two-dimensional grid. We will apply two-dimensional CNNs in Section 3.4.3.

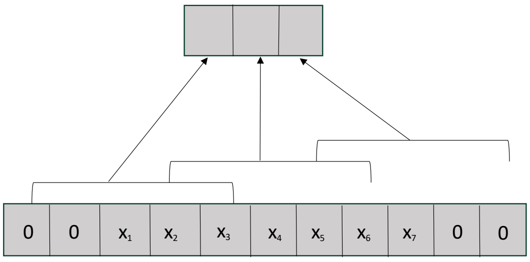


Figure 1.2: CNN with kernel size 5, stride 3 and padding 2.

1.2.2 Activation functions

The *activation function* is one of the most important hyperparameters in NN architectures. The purpose of this hyperparameter is to introduce nonlinearity into the prediction. There are many activation functions proposed in the literature; see the comprehensive survey [46] for more details. However, there is no basic rule for the appropriate choice of the activation function.

The most commonly used activation functions are: *Rectified linear unit* (RELU), *Exponential linear unit* (ELU), *Sigmoid*, *Hyperbolic tangent* (Tanh) or *Softplus*. With the RELU activation function, negative inputs are set to zero and positive inputs remain the same. It is widely used because of its simplicity. However, it can cause a *dying RELU problem*, which is a situation where certain neurons become inactive during training and never recover. This is caused by the zero gradient for all negative input values. ELU is a differentiable activation function that can avoid the problem of RELU. For negative values, non-zero output is obtained. Sigmoid ensures that the output of the NN is between 0 and 1. Similarly, Tanh maps the output of the NN to a range $[-1, 1]$. However, both of these functions can cause a *vanishing gradient problem*. Their derivative becomes very small for large positive or negative inputs, resulting in vanishing gradients during the backpropagation. This is because these functions saturate, flatten out and lead to gradients close to zero. The Softplus activation function ensures that the output is always positive. Unlike RELU, it is smooth and continuous.

1.2.3 Loss function

The loss function is critical in training deep learning models because it quantifies the difference between predicted and target values, guiding the optimization process towards better performance. Commonly used is the loss function based on the L_2 norm, which offers the benefit of stronger gradients, leading to faster training.

1.2.4 Optimizer and the optimal learning rate

Let us first introduce the *Gradient descent algorithm.* Gradient descent is an optimization algorithm used to find a minimum of a given function by iteratively moving in the direction of its steepest descent. In each iteration, the gradient of a function with respect to the parameters is computed and scaled by a weighting factor (*learning rate*). After that, the parameters of a function are updated. This process continues until convergence or a stopping criterion is met, effectively finding the minimum of a function.

In machine learning, a very common algorithm is *Stochastic gradient descent* (SGD). In SGD, instead of computing gradients on the entire dataset, gradients are computed on small random subsets or individual data points in each iteration. This introduces randomness and noise into the gradient estimation process, but makes SGD faster and more scalable. In addition, SGD with momentum uses the gradients from the previous update steps to speed up the gradient descent [92].

Even more popular is the Adam optimizer [55], which is a variant of previously developed optimizers *AdaGrad* [22] and *RMSProp* [44]. It dynamically tunes the learning rate, ensuring it neither grows excessively large nor diminishes too quickly. It combines the benefits of previously developed algorithms, helping the model converge faster and more reliably.

The learning rate is another important hyperparameter to choose. A larger learning rate may miss the local minima, and a smaller learning rate may require a large number of iterations to reach convergence. Therefore, it is important to find a near-optimal learning rate. Finding the near-optimal learning rate typically requires experimentation.

1.2.5 Backpropagation algorithm

Having described the different NN structures, the role of the loss function, the activation functions and the optimizer, this section explains how the weights of the NN are actually updated.

After the forward pass of the training algorithm (the process of calculating the output of an NN), the error (loss function) is calculated. Then, going backward through each layer, the error contribution of each connection must be measured. This is done by *backpropagation algorithm.* Starting from the output layer, the gradient of the loss function is calculated with respect to the parameters (weights and biases) of each neuron. Then the successive application of the chain rule is used to compute the gradients, which are used to update the weights and biases using optimization algorithms to minimize the loss function.

1.2.6 Possible improvements and modifications

There are many ways to modify the developed NN. In this thesis we experiment with the *residual connection framework* [41] to improve the gradient propagation to the lower layers of the NN. The idea is to introduce a so-called *identity mapping*, which simply adds the output of the previous layer to the output of the next layer. It is important that no additional parameters or computational complexity are added. This framework will be used in Chapter 7.

Another possible modification is the use of *adaptive activation functions.* We use them in Chapters 6 and 8. They can dynamically adjust their parameters or shape based on the input data. More details can be found in Section 6.1.1.

1.2.7 Combination of numerical methods with deep learning

In this thesis we combine the classical numerical methods with deep learning by inserting NN parts to them. The whole numerical method can be then seen as a large NN. Although we optimize only the parameters of the small embedded NN, we need to backpropagate the gradients also through the numerical method until we reach them. This can be done, as the backpropagation is based on the successive use of the chain rule to compute derivatives of arbitrarily complicated functions, which can be represented by a directed acyclic graph (DAG), where each node represents a primitive operation such as $+$, $-$, min or max with defined derivatives. Note that we do not need to implement the backpropagation algorithm by ourselves. It is enough to implement a numerical method in an established framework such as *Pytorch* [79] (`https://pytorch.org/`) and make sure that all operations we are using are differentiable.

1.3 Outline of the thesis

Let us now present the structure of the Thesis:

Chapter 1: In this chapter we introduce numerical mathematics and deep learning. We introduce basic NN structures, activation functions, explain the role of a loss function, optimizer and learning rate. Finally, we describe the backpropagation algorithm and how it can be used in combination with various numerical methods. We also propose two modifications that can be used to improve the training procedure.

Chapter 2: In Chapter 2 we provide the detailed literature overview of useful literature, where machine learning has been successfully applied to various numerical algorithms. Two main approaches are mentioned. First, where machine learning techniques are used to directly approximate the solution of a given PDE. Second, where machine learning is used to improve standard numerical methods. In the second case, the output of the learning procedure is the entire numerical scheme.

Chapter 3: This chapter is based on the papers [60] and [61]. We introduce the standard FDM and the compact FDM. We then describe the algorithm based on the approximation of the discretization error by deep learning. We present the training procedure and demonstrate the performance of the developed method on several examples. We also provide the reader with a time complexity analysis to demonstrate the efficiency of the method. We present this approach as a proof of concept that deep learning can be effectively combined with the standard numerical schemes to achieve better results.

Chapter 4: In Chapter 4 we provide the mathematical background for hyperbolic conservation laws. We introduce the numerical methods for solving these problems, in particular the construction of conservative numerical schemes. Later in this chapter, we explain the motivation for using the WENO scheme to solve hyperbolic conservation laws and nonlinear degenerate parabolic equations. We describe the development of the WENO method and provide the reader with a detailed literature overview for various WENO schemes and related methods.

Chapter 5: In this chapter, based on the paper [57], we first describe the WENO algorithm developed for solving hyperbolic conservation laws [50]. We explain the role of the smoothness indicators and introduce the modification of the standard WENO scheme, namely the WENO-Z scheme [13]. Next, we present our deep learning approach using CNNs to further improve the WENO methods, called WENO-DS, without any post-processing. Later in the chapter, we provide the corresponding proofs of the formal order of accuracy for two WENO schemes. After describing the two-dimensional implementation, we present our numerical results, which illustrate the improvements of our proposed method.

Chapter 6: This chapter is based on the papers [57] and [62] and we introduce the one- and two-dimensional Euler system of gas dynamics. We describe the application of the proposed deep learning based WENO-DS method for solving this system and introduce a novel training procedure. We then present in detail the numerical results with a wide range of test configurations. We consider different types of CNNs and discuss their advantages and disadvantages.

Chapter 7: In this chapter, based on the paper [59], we present the general framework of the WENO methods for solving nonlinear degenerate parabolic equations [35, 74]. We then explain how the smoothness indicators are modified using the deep learning algorithm. We provide a detailed proof of the formal sixth-order accuracy of the novel WENO-DS scheme, describe the structure of the CNN considered in this application, and explain the training procedure. We also explain how we proceed for two-dimensional problems. Finally, we present the results on numerous numerical examples, where we demonstrate our improvements with figures and tables.

Chapter 8: This chapter is based on paper [58]. We present the novel WENO-DS method and its application in finance. First, we introduce the Black-Scholes equation and we consider a European digital option as an illustrating example. Here we avoid the undesirable oscillations, especially in the first time steps of the numerical solution.

Chapter 9: In the last chapter we conclude our work and offer insights into potential directions of the future research.

Part I

Enhanced finite difference methods by deep learning

2 Chapter 2

Introduction to deep learning based numerical algorithms and literature overview

In recent years, there has been an increased interest in solving PDEs using deep learning, see e.g. [10, 12, 25, 53, 84, 85, 106]. 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 [10] for further information on deep neural networks (DNNs) methods for solving PDEs in high-dimensions. On the other hand, DNN-based PDE solvers generally can not compete with classical numerical solution techniques in lower dimensions - since solving the highly nonlinear optimization problems in the training phase is too costly.

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 [103] 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 [23] presented the *Deep Ritz Method* (DRM) for the numerical solution of variational problems in high dimensions. Also, He et al. [40] theoretically analyzed the relationship between DNN with RELU function as the activation function and the finite element method (FEM). For the proper treatment of the boundary conditions, see [76].

Recently, machine learning was widely used to compute the solution of PDEs. We refer to [11, 65, 103], where the NN algorithm is used to approximate a solution of a particular PDE problem. Following that approach, the solution of a particular PDE is a result of a NN training procedure. Another idea is to improve a specific numerical scheme using NNs. The training of a NN is made offline and results in a new numerical scheme applicable to a wider class of problems. This idea was recently used by Beck et al. [9] for discontinuous Galerkin methods or in [45] for learning iterative PDE solvers. Bar-Sinai et al. [7] use NNs and learn from high resolution solutions to approximate a spatial derivative on a coarse grid. We refer the reader also to [20, 26, 89] for other work in this direction.

In 2019, Raissi, Perdikaris, and Karniadakis [85] 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 NN 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 [86] 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 FDMs 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 [33].

In [80], a graph neural networks based framework called *MeshGraphNets* was used. This efficient approach is intended for learning mesh-based high-dimensional scientific simulations and includes adaptive mesh discretization. Trask et al. [108] generalized CNNs for data on unstructured stencils based on *Generalized Moving Least Squares* (GMLS). In [93], a so-called *Deep Learning Discrete Calculus* (DLDC) was proposed, which uses the knowledge from discrete numerical methods, such as FDMs and FEMs, to interpret the deep learning algorithms.

In recent years, deep learning has been used not only to solve certain PDEs directly, but also to improve existing numerical methods. This motivates us to propose new FDMs in combination with DNNs. We refer to the new method as the deep finite difference method (DFDM). 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. 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 [96] 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 [52, 118] for related hypersolver approaches. The basic idea of this method is to augment an ODE solver with a NN 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 [96], which we briefly summarize in the sequel. First, unlike [96], we use the idea of approximating the local truncation error for solving PDEs rather than ODEs. Moreover, we use a different NN structure, namely a very small CNN, to ensure time efficiency. In [96], a multi-layer fully connected NN with 8 layers and 80 neurons is used. In our approach, the NN 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 [96], the NN 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 [96], the input to the NN 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 NN itself. By using CNN, the spatial neighborhood from the previous time step is also part of the input.

The main advantages of our proposed scheme are that the scheme remains convergent and consistent. Although we improve the standard FDM and compact FDM, this approach can be easily extended to any traditional numerical scheme, for which a truncation error is available. The method is straightforward and 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 NN.

3

Chapter 3

Deep finite difference method

In this chapter we propose a new idea to improve numerical methods for solving 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 FDMs, but it can be easily generalized to other numerical methods, for which a truncation error is available.

Without losing the consistency and convergence of the FDM, 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.

3.1 Standard finite difference method

Let us consider a (parabolic) PDE of the form

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \sum_{p,r=1}^{d} \alpha_{pr}(\mathbf{x}) \frac{\partial^2 u}{\partial x_p \partial x_r} + \sum_{p=1}^{d} \beta_p(\mathbf{x}) \frac{\partial u}{\partial x_p} + \gamma(\mathbf{x})u, \quad (\mathbf{x}, t) \in \Omega_d \times [0, T], \\
&u(\mathbf{x}, 0) = u_0(\mathbf{x}),
\end{aligned} \tag{3.1}$$

with the coefficients $\alpha_{pr}, \beta_p, \gamma \colon \Omega_d \subseteq \mathbb{R}^d \to \mathbb{R}$, $p, r = 1, \dots, d$, where d denotes the space dimension and $\mathbf{x} = (x_1, \dots, x_d) \in \Omega_d$. We start with the simple one-dimensional case where the PDE (3.1) reduces to

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \alpha(x) \frac{\partial^2 u}{\partial x^2} + \beta(x) \frac{\partial u}{\partial x} + \gamma(x)u, \qquad (x, t) \in \Omega_1 \times [0, T], \\
&u(x, 0) = u_0(x).
\end{aligned} \tag{3.2}$$

We select the one-dimensional 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 differ-*

ence method. The well-known second order central approximation to the second derivative is given by

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} = \frac{u(x_{i+1},t) - 2u(x_i,t) + u(x_{i-1},t)}{\Delta x^2} - \frac{\Delta x^2}{12}\left.\frac{\partial^4 u}{\partial x^4}\right|_{x_i} + O(\Delta x^3), \tag{3.3}$$

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

$$\left.\frac{\partial u}{\partial x}\right|_{x_i} = \frac{u(x_{i+1},t) - u(x_{i-1},t)}{2\Delta x} - \frac{\Delta x^2}{6}\left.\frac{\partial^3 u}{\partial x^3}\right|_{x_i} + O(\Delta x^3), \tag{3.4}$$

for $u \in C^3(\Omega_1)$. Let us use the short notation

$$u^{(2)}(x_i) = \left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i}, \qquad u^{(1)}(x_i) = \left.\frac{\partial u}{\partial x}\right|_{x_i} \tag{3.5}$$

and define the local discretization error of the finite difference scheme.

Definition 1. *Let us denote the exact solution at grid point x_i as u_i and the exact value of the derivative at the same point as $u^{(l)}(x_i)$, where l denotes the order of the derivative. Let $L(u_i, x_i, \Delta x)$ represent the discretized version of the derivative at grid point x_i using a specific finite difference scheme with grid spacing Δx. The local discretization error $\epsilon_i^{(l)}$ at grid point x_i is then given by*

$$\epsilon_i^{(l)} = L(u_i, x_i, \Delta x) - u^{(l)}(x_i). \tag{3.6}$$

It can be seen, that the local discretization error $\epsilon_i^{(2)} = O(\Delta x^2)$ and $\epsilon_i^{(1)} = O(\Delta x^2)$ is of the second order for both schemes (3.3) and (3.4), 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 NN trained to approximate the local discretization error $\epsilon_i^{(1)}$ and $\epsilon_i^{(2)}$ such that the final numerical approximation $\hat{u}_i^n$ is improved. Let us remind the reader that we abbreviate our resulting new deep learning finite difference method as DFDM. The further details of this method will be discussed in Section 3.1.1.

3.1.1 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 in our application the 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 of the CNN.

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 (3.2) using (3.3), (3.4) and adding the NN function terms $F(\hat{u}^n)_i, G(\hat{u}^n)_i$. This leads to the following deep FDM ansatz

$$\begin{aligned} \hat{u}_i^{n+1} = \hat{u}_i^n + \Delta t\Big[\alpha(x_i)\Big(\frac{\hat{u}_{i+1}^n - 2\hat{u}_i^n + \hat{u}_{i-1}^n}{\Delta x^2} + \Delta x^2 F(\hat{u}^n)_i\Big) \\ + \beta(x_i)\Big(\frac{\hat{u}_{i+1}^n - \hat{u}_{i-1}^n}{2\Delta x} + \Delta x^2 G(\hat{u}^n)_i\Big) + \gamma(x_i)\,\hat{u}_i^n\Big], \end{aligned} \tag{3.7}$$

where for NN functions holds $F(\hat{u}^n)_i = F(\bar{u}_i^n)$ and $G(\hat{u}^n)_i = G(\bar{u}_i^n)$ with $\bar{u}_i^n = \bar{u}^n(\bar{x}_i) = (\hat{u}^n(x_{i-k}), \ldots, \hat{u}^n(x_{i+k})) = (\hat{u}_{i-k}^n, \ldots, \hat{u}_{i+k}^n)$ being the input to the NN. This means, that we feed the CNN with the previously computed numerical approximations. When applying a CNN kernel to compute $F(\bar{u}_i^n)$ and $G(\bar{u}_i^n)$, under the receptive field 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 receptive field 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:

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

and

$$F(\bar{u}_i^n) = G(\bar{u}_i^n) = O(1), \qquad \text{for} \qquad \Delta x \to 0.$$

The convergence and consistency properties of the standard FDM are preserved. This is ensured due to multiplication of the NN functions with the step size Δx^2 as in (3.7). Moreover, the values of the NN 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 (3.3), (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 Section 3.2.

3.2 Compact finite difference method

Let us consider as benchmark a heat equation of the form

$$\begin{aligned} &\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2} \qquad (x,t) \in \Omega_1 \times [0,T], \\ &u(x,0) = u_0(x), \end{aligned} \tag{3.8}$$

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. [68]. For example, the second derivative can be implicitly computed using the fourth-order compact scheme

$$\frac{1}{10}u''_{i+1} + u''_i + \frac{1}{10}u''_{i-1} = \frac{1}{\Delta x^2}\Big(\frac{6}{5}u_{i+1} - \frac{12}{5}u_i + \frac{6}{5}u_{i-1}\Big) + O(\Delta x^4). \tag{3.9}$$

Here, the discretization error fulfills $\epsilon_i^{(2)} = O(\Delta x^4)$ for $u \in C^6(\Omega_1)$.

3.2.1 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 with an available truncation error. We again consider the CNN and add properly the NN function term to the discretization of the PDE (3.8). Here, we use for the time discretization the trapezoidal rule, which is second order in time:

$$\frac{\hat{u}^{n+1} - \hat{u}^n}{\Delta t} = \frac{1}{2}\alpha\Big(\hat{u}''^{n+1} + \hat{u}''^{n}\Big) + \Delta x^4 F(\hat{u}^n), \tag{3.10}$$

where $F(\hat{u}^n)$ is a vector with elements $F(\hat{u}^n)_i = F(\bar{u}_i^n)$ with $\bar{u}_i^n = \bar{u}^n(\bar{x}_i) = (\hat{u}^n(x_{i-k}), \dots, \hat{u}^n(x_{i+k})) = (\hat{u}_{i-k}^n, \dots, \hat{u}_{i+k}^n)$ 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 (3.9) 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

$$2\hat{u}^{n+1} - \alpha\Delta t A^{-1}(B\hat{u}^{n+1} + c) = 2\hat{u}^n + \alpha\Delta t A^{-1}(B\hat{u}^n + d) + 2\Delta x^4 \Delta t F(\hat{u}^n), \tag{3.11}$$

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

$$(2A - \alpha\Delta t B)\hat{u}^{n+1} = (2A + \alpha\Delta t B)\hat{u}^n + \alpha\Delta t(c + d) + 2A\Delta x^4 \Delta t F(\hat{u}^n). \tag{3.12}$$

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

$$F(\bar{u}_i^n) \approx \frac{1}{\Delta x^4}\,\epsilon_i^{(2)} \qquad \text{and} \qquad F(\bar{u}_i^n) = O(1), \qquad \text{for} \qquad \Delta x \to 0.$$

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

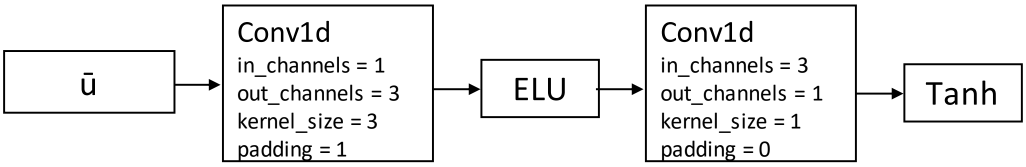
3.3 Training procedure

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 3.1. This small NN 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 NN to compute the functions $F(\hat{u}^n)$ and $G(\hat{u}^n)$ from (3.7). These are then represented as two output channels of the NN, 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.

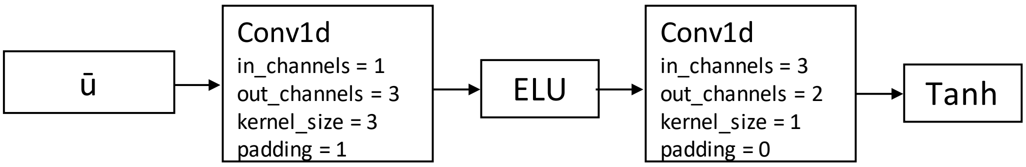
At the beginning of the training procedure, the weights of the CNN are initialized randomly. Then, a problem is randomly selected from the dataset. The discrete computational domain is divided into $I \times N$ steps ($I \times J \times N$ for two-dimensional problems), where I, J are the number of space 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 (3.7) is the input to the CNN. For the optimization of the loss function we use the Adam optimizer [55], where the learning rate is set separately for each PDE class. For the training procedure, we use the mean squared error loss function, defined as

$$\mathrm{LOSS}_{\mathrm{MSE}}(u) = \frac{1}{I}\sum_{i=0}^{I}(\hat{u}_i^n - u_i^{n,\mathrm{ref}})^2, \tag{3.13}$$

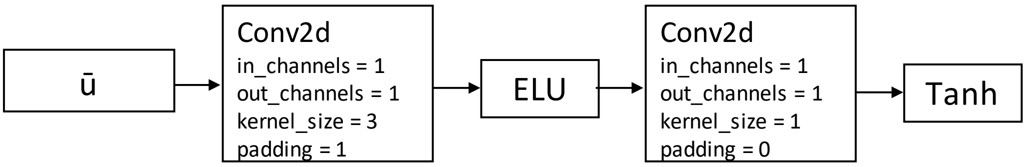
where $\hat{u}_i^n$ is a numerical approximation of $u(x_i, t_n)$ obtained by DFDM and DCFDM, respectively, and $u_i^{n,\mathrm{ref}}$ denotes the reference solution. If the exact solution is available, this is used as the reference solution. Otherwise, the reference solution is calculated on a very fine grid. For the implementation we use Python with the library PyTorch [79]. We summarize the training procedure and the implementation of DFDM in Algorithm 1. The training procedure results in a new numerical



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



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



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

Figure 3.1: Structure of the CNN for different examples.

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 (3.7), resp. (3.12) and compute the solution approximation $\hat{u}^{n+1}$ at time $t_{n+1}$
        ◇ compute loss using equation (3.13)
        ◇ compute loss gradient with respects to the weights of CNN
        ◇ update weights using chosen optimizer
    end for
    ◇ testing on validation problems
end for
```

3.4 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 in 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).

3.4.1 One-dimensional heat equation

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

$$u_t = u_{xx}, \quad u(x,0) = c + a\sin(b\pi x), \quad -\pi \le x \le \pi, \quad 0 \le t \le T. \tag{3.14}$$

The exact solution for this example is

$$u(x,t) = c + a\, e^{(-b^2\pi^2 t)} \sin(b\pi x) \tag{3.15}$$

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

We proceed during the training as described in Section 3.3 and specify the learning rate for the Adam optimizer as $\mathrm{lr} = 0.00001$. 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

$$a \in \mathcal{U}[1,2], \quad b \in \mathcal{U}[0.3, 0.5], \quad c \in \mathcal{U}[0, 0.25]. \tag{3.16}$$

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 3.2. 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 (3.13) for each validation problem to be in the interval $[0,1]$ using the relation

$$\mathrm{LOSS}^{*}_{\mathrm{MSE}}(u) = \frac{\mathrm{LOSS}^{l}_{\mathrm{MSE}}(u)}{\max\limits_{l=0,\dots,L}(\mathrm{LOSS}^{l}_{\mathrm{MSE}}(u))}, \qquad l = 0, \dots, L, \tag{3.17}$$

where L denotes the total number of training cycles.

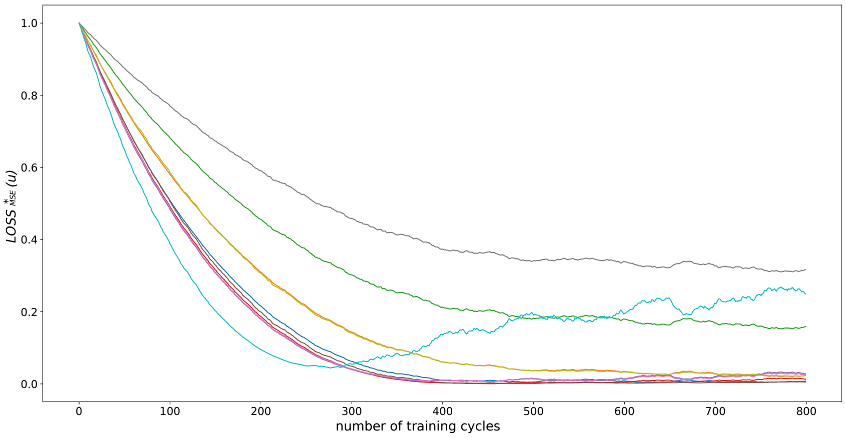


Figure 3.2: Training evolution corresponding to one-dimensional heat equation: The values (3.17) for different validation problems evaluated after each training cycle.

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) $T = 0.25$

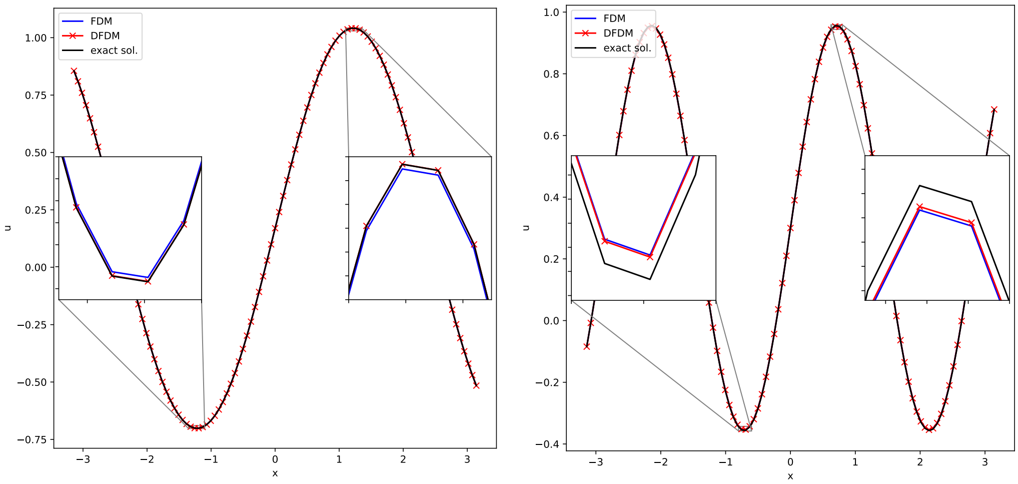
parameters			L_∞			L_2		
a	b	c	FDM	DFDM	ratio	FDM	DFDM	ratio
1.88	0.32	0.12	**0.000380**	0.000577	0.66	**0.000673**	0.000956	0.70
1.31	0.4	0.12	0.000496	**0.000072**	6.89	0.000817	**0.000107**	7.65
1.15	0.43	0.15	0.000515	**0.000082**	6.28	0.000816	**0.000121**	6.76
1.95	0.42	0.16	0.000828	**0.000175**	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 3.1: Comparison of L_∞ and L_2 errors of FDM and DFDM for the solution of the heat equation (3.14) with various parameters a, b, c and T, $I = 100$.

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 a model obtained after the 685th 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 3.3 illustrates the solution for two different initial value parameters choices. We observe, that the method performs well also on the set of parameters a, b, c outside of the training interval. In Table 3.1 we can see the significant improvement on the errors. Although we used fixed final time $T = 0.25$ during training, we see, that the method performs very well also with the final time $T = 0.5$, where the improvement ratio reaches high values in many cases.



(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.)

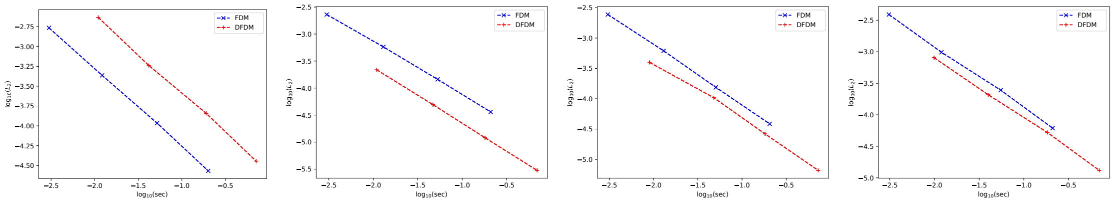
Figure 3.3: Comparison of the FDM and DFDM for the solution of the heat equation (3.14), $I = 100$, $T = 0.25$.

Furthermore, we analyze the computational cost of our method and compare it in Figure 3.4. 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.

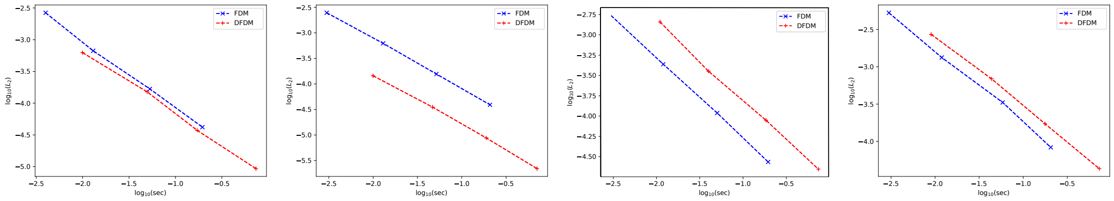
Next, we retrain the NN for the following diffusion-convection equation

$$u_t = \alpha u_{xx} - \beta u_x, \quad u(x, 0) = c + a\sin(b\pi x), \quad 0 \le x \le 2\pi, \quad 0 \le t \le T, \tag{3.18}$$

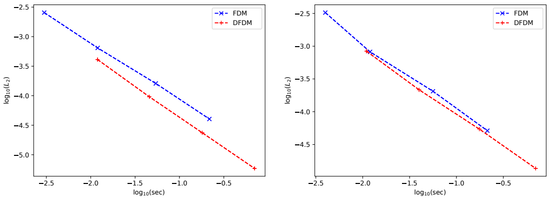
where in addition to parameters from (3.16) also the parameters $\alpha \in \mathcal{U}[1, 2]$ and $\beta \in \mathcal{U}[1, 2]$ are chosen randomly during the training and testing. The training is



(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 3.4: Comparison of computational cost against L_2 error on the solution of the heat equation (3.14) with various parameters a, b and c according to Table 3.1. $T = 0.25$.

performed as described before and we choose the learning rate $\mathrm{lr} = 0.0001$. The CNN structure can be found in Figure 3.1b and we run the training for 4000 training cycles. We present in Table 3.2 the results for different parametrizations of the PDE and the initial condition (3.18). We see, that we obtain in all cases smaller errors using DFDM.

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 3.2: Comparison of L_∞ and L_2 errors of FDM and DFDM for the solution of the diffusion-convection equation (3.18) with various parameters α, β, a, b, c, $I = 100$, $T = 0.25$.

3.4.2 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

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV = 0, \quad S \in [0,\infty),\ t \in [0,T], \tag{3.19}$$

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 thesis, we solve the European call option pricing problem with the following terminal and boundary conditions:

$$\begin{aligned} &V(S,T) = \max\{0, S-K\} =: (S-K)^+, \\ &V(S,t) \to 0, \quad \text{for} \quad S \to 0, \qquad V(S,t) \to S - Ke^{-r(T-t)}, \quad \text{for} \quad S \to \infty, \end{aligned} \tag{3.20}$$

with K being the strike price. We use the following transformation of variables that exploits the Euler structure of the spatial operator in (3.19) and also reverses the time direction:

$$S = Ke^x, \quad \tau = T - t, \quad V(S,t) = Ku(x,\tau) \tag{3.21}$$

and substitute this into (3.19) and (3.20). Then we obtain the (forward-in-time) PDE:

$$u_\tau = \frac{\sigma^2}{2}u_{xx} + \left(r - \frac{\sigma^2}{2}\right)u_x - ru, \quad x \in \mathbb{R},\ 0 \le \tau \le T. \tag{3.22}$$

For the training, we generate randomly the parameters

$$\sigma \in \mathcal{U}[0.4, 0.6], \quad r \in \mathcal{U}[0.1, 0.3]. \tag{3.23}$$

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 $\Delta\tau = 0.8\Delta x^2/\sigma^2$.

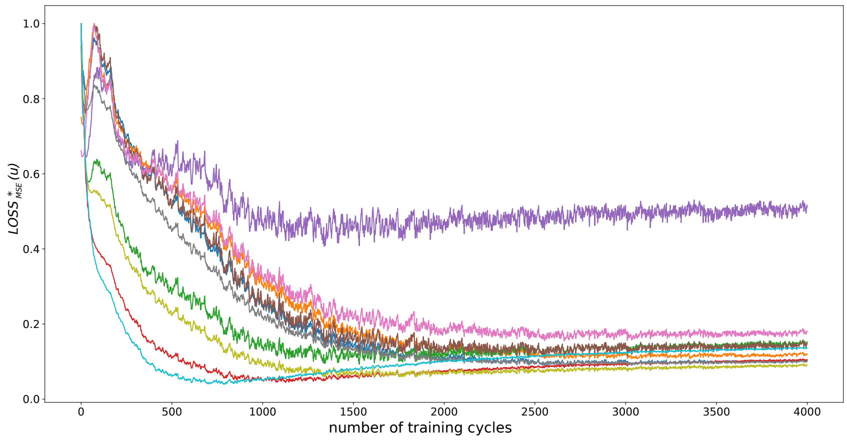


Figure 3.5: Training evolution corresponding to European call option example: The values (3.17) for different validation problems evaluated after each training cycle.

During the training we use the NN structure as in Figure 3.1b. We use the learning rate lr = 0.0001 and run the training for 4000 training cycles with fixed final time $T = 1$. Figure 3.5 shows the evolution of the rescaled values (3.17). Using the same decision rule for the best model as in example from Section 3.4.1 we choose the model obtained after the 1532nd training cycle. Numerical results on problems from the test set can be found in Table 3.3. Similarly to previous examples, also in this test case performs DFDM significantly better.

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 3.3: Comparison of L_∞ and L_2 errors of FDM and DFDM for the solution of the Black-Scholes equation (3.19) with various parameters σ and r, $I = 50$, $T = 1$.

3.4.3 Two-dimensional heat equation

Here we extend the example from Section 3.4.1 to two space dimensions. We solve the following two-dimensional heat equation

$$\begin{gathered} u_t = u_{xx} + u_{yy}, \\ u(x,0) = c + a\sin(b\pi x) + d\sin(e\pi y), \quad -\pi \le x, y \le \pi, \quad 0 \le t \le T. \end{gathered} \tag{3.24}$$

For the 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 3.1c. 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 in the one-dimensional example in Section 3.4.1 and we run it for 3000 training cycles. We again choose the model with the best performance on the validation set as described in Section 3.4.1 and present the numerical results on problems from the test set in Table 3.4 and in Figure 3.6. We see, that the results with fixed final time $T = 0.25$ as well as with $T = 0.5$ are significantly better in all cases.

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 3.4: Comparison of L_∞ and L_2 errors of FDM and DFDM for the solution of two-dimensional heat equation (3.24) with various parameters a, b, c, d, e and T. $I \times J = 50 \times 50$.

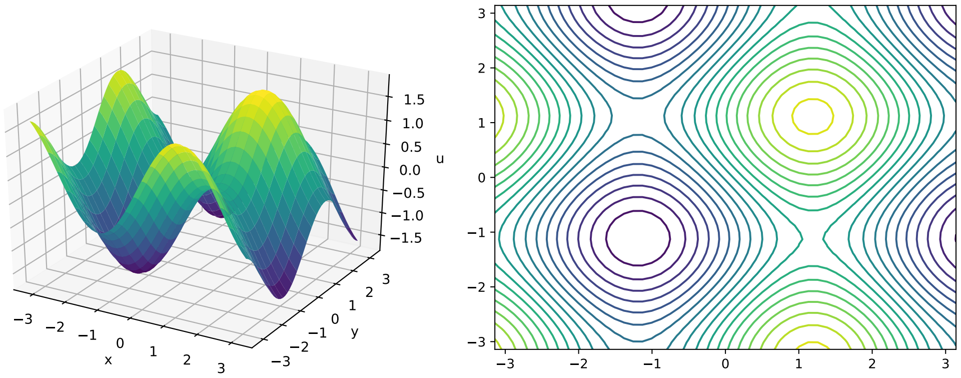


Figure 3.6: Solution of two-dimensional heat equation (3.24) with parameters $a = 1.7$, $b = 0.42$, $c = 0.05$, $d = 1.2$, $e = 0.45$. $I \times J = 50 \times 50$, $T = 0.25$.

3.4.4 One-dimensional heat equation with Deep CFDM

In the next example we apply the DCFDM to the one-dimensional heat equation

$$u_t = \alpha u_{xx}, \quad u(x,0) = c + a\sin(b\pi x), \quad -\pi \le x \le \pi, \quad 0 \le t \le T. \tag{3.25}$$

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

$$\alpha \in \mathcal{U}[1,2], \quad a \in \mathcal{U}[1,2], \quad b \in \mathcal{U}[0.3,0.5], \quad c \in \mathcal{U}[0,0.25]. \tag{3.26}$$

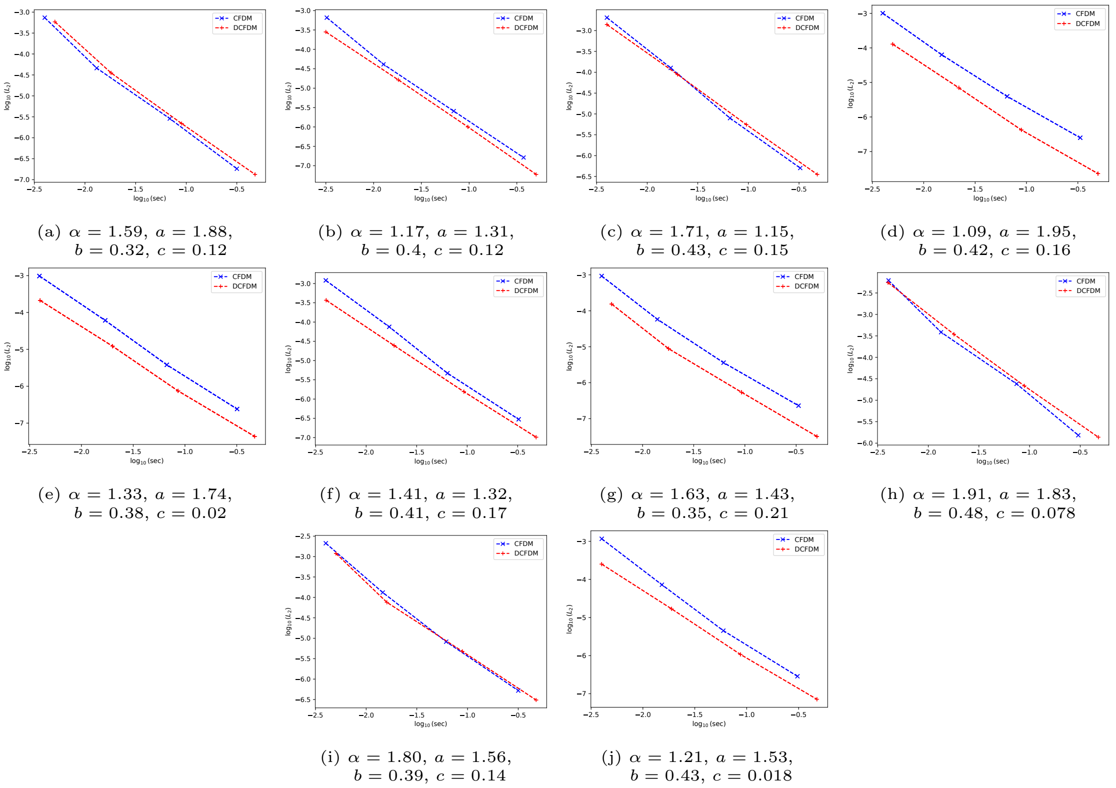
Further, for the training we set $T = 0.25$, divide the computational domain into 25 space steps and for the temporal step size use $\Delta t = \Delta x^2$, i.e. $\lambda = 1$. The NN structure is used as in Figure 3.1a. We select the learning rate $\mathrm{lr} = 0.0001$ and run the training for 2000 training cycles. We choose the final model according to the rule in the example from Section 3.4.1 and present the results in Table 3.5. We see, that the NN can improve the CFDM very well. The improvement ratios are slightly smaller when we compare the results with Table 3.1. However, 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 original CFDM. As illustrated in Figure 3.7, the DCFDM remains time effective in most of the cases. We note, that we did not retrain the method for different spatial discretizations.

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 3.5: Comparison of L_∞ and L_2 errors of CFDM and DCFDM for the solution of the heat equation (3.25) with various parameters α, a, b, c, $T = 0.25$, $I = 50$.

3.5 Asian option pricing problem and modification of Deep FDM

The next example from computational finance is the problem of pricing Asian options. Asian options belong to the class of so-called path-dependent options. These options have a characteristic payoff that depends on the average price of an underlying asset, e.g. the asset S. There are many types of Asian options. The average



(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 3.7: Comparison of computational cost against L_2 error on the solution of the heat equation (3.25) with various parameters α, a, b and c according to Table 3.5, $T = 0.25$.

can be arithmetic or geometric, and it can be determined by the discrete or continuous prices of an asset. In this case too, there are Asian options of the European or American type.

In this thesis we consider Asian options of the European type depending on the arithmetic mean of the asset price and we take the continuous case. In addition, there are four different types of arithmetic Asian option with respect to the payoff function, and we consider the fixed-strike call option. In this case, we obtain the equation for the price of the Asian option, which is given by the two-dimensional PDE. More details can be found e.g. in [56].

To avoid solving the two-dimensional PDE, Rogers and Shi [90] introduced a reduced PDE of the form

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2} - \left(\frac{1}{T} + rx\right)\frac{\partial u}{\partial x} = 0, \quad 0 \le t \le T, \quad x \in \mathbb{R}, \tag{3.27}$$

which can be solved equivalently for European style of Asian options. Here, r denotes an interest rate, σ the volatility, x is defined by

$$x = \frac{1}{S_t}\Big(K - \int_0^t S_v\, \mu(dv)\Big), \tag{3.28}$$

where μ is the probability measure with the density $\rho(t) = 1/T$ in our case. Further, S is the price of an underlying asset at time t and K the strike price. The price of an option V is then given by $V = S_0 u(K/S_0, 0)$ for some initial stock price S_0.

After transforming (3.27) to the forward-in-time PDE using $\tau = T - t$, we obtain

$$\frac{\partial u}{\partial \tau} = \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2} - \left(\frac{1}{T} + rx\right)\frac{\partial u}{\partial x}, \quad 0 \leq \tau \leq T, \quad x \in \mathbb{R}, \tag{3.29}$$

which has a form of the PDE (3.2). We use the initial condition [90]

$$u(x, 0) = \max(0, -x), \tag{3.30}$$

and the boundary conditions [90]

$$u_0 = \frac{1 - e^{-r\tau}}{rT} - x_0 e^{-r\tau}, \qquad u_N = 0. \tag{3.31}$$

We select the computational domain $[x_l, x_r] = [-0.4, 4]$ and final time $T = 1$.

3.5.1 Modification of the scheme and training procedure

The coefficient in front of a convection term $(\frac{1}{T} + rx)$ is always positive in our case and can become dominant. This means that to overcome the oscillations, which could appear in the solution, the left-biased stencil should be used to approximate the first derivative. This reads

$$\left.\frac{\partial u}{\partial x}\right|_{x_i} = \frac{u(x_i, t) - u(x_{i-1}, t)}{\Delta x} + O(\Delta x). \tag{3.32}$$

As we can see, the approximation is only of the first order, so for the enhanced DFDM we can use the formula

$$\hat{u}_i'^n = \frac{\hat{u}_i^n - \hat{u}_{i-1}^n}{\Delta x} + \Delta x G(\bar{u}_i^n), \tag{3.33}$$

to get the deep learning improved approximation of the first derivative. Now we insert the equation (3.33) into (3.7), replacing the central approximation of the first derivative, and use it as our final scheme.

Remark 1. Also one-sided second order finite difference approximation for the first derivative could be used. However, for this we would need to define one more boundary point on the left side of the computational domain. Moreover, wider stencil can again lead to numerical oscillations. In Section 3.5.2 we will compare the numerical results using (3.33) as well as using second-order finite difference approximations for the first derivative.

We also introduce a multiplication factor for the deep learning terms

$$\begin{aligned} F(\bar{u}_i^n) &= 10^3 \min\big(|\hat{u}_{i+1}^n - 2\hat{u}_i^n + \hat{u}_{i-1}^n|, 10^2\big)\, \mathcal{F}(\bar{u}_i^n), \\ G(\bar{u}_i^n) &= 10^2 \min\big(|\hat{u}_{i+1}^n - 2\hat{u}_i^n + \hat{u}_{i-1}^n|, 10^2\big)\, \mathcal{G}(\bar{u}_i^n). \end{aligned} \tag{3.34}$$

By adding these factors, we ensure that the bounded learned coefficients have significant effects only in the non-smooth (kinked) part of the solution. By ensuring that these factors are bounded, we do not destroy the convergence properties.

For computational efficiency, we use a small CNN structure described in Figure 3.1b. Two output channels in the last hidden layer represent the correction $\mathcal{F}(\bar{u}_i^n)$ of a diffusion term and the correction $\mathcal{G}(\bar{u}_i^n)$ of a convection term of (3.29).

We aim to obtain a numerical scheme, which can reliably solve the Asian option pricing problem for all possible combinations of σ, r and T. For this purpose, we first create a data set consisting of 100 reference solutions. We generate the parameters σ and r randomly

$$\sigma \in \mathcal{U}[0.1, 0.4], \qquad r \in \mathcal{U}[0.1, 0.3]. \tag{3.35}$$

For training, we fix $T = 1$, $K = 100$ and use the computational domain $[x_l, x_r] = [-0.4, 4]$. The reference solutions are computed using standard central finite difference schemes on a grid divided into 400 space points and the temporal step size is chosen such that $N = \max\limits_{i=0,\dots,I} \Big((\tau\sigma^2 x_i^2)/(0.8\Delta x^2)\Big)$.

For the training, we use the training procedure described in Section 3.3 We divide the spatial computational domain into 50 space steps. Then, we randomly select a problem from a data set. Afterwards, we compute successively the solution up to the fixed final time T. After each time step n we compute the loss with respect to the weights of CNN, update the weights and continue to the next time step $n + 1$. This means, in each subsequent time step, a new updated solution according to (3.7) is input to the CNN. For the optimization we use the Adam optimizer with the learning rate 0.0001. For the training procedure, we use the mean squared error loss function (3.13).

After the training, we choose the model from a training step, in which the best performance on problems from the validation set is obtained. These are the problems with randomly generated initial parameters, which were not in the training data. This is our final DFDM and we present the numerical results using this method in the following section.

3.5.2 Numerical results

Let us present the numerical results on a test set containing the problems with the randomly generated initial data.

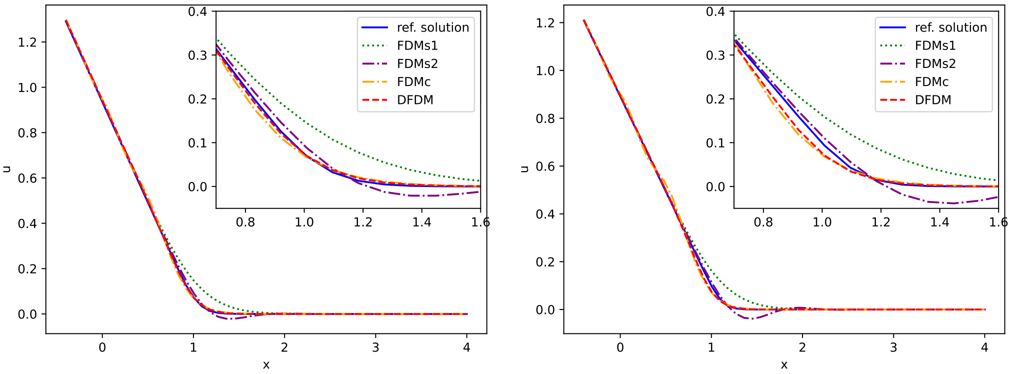
We compare the L_2 errors in Table 3.6. We computed the solution with the given r and σ parameters as given in the table. For the computation, we used the central finite difference formula for the diffusion term, and for the convection term, we distinguish among the following possibilities: second-order central finite difference scheme (FDMc), second-order one-sided finite difference scheme (FDMs2), first-order one-sided finite difference scheme (FDMs1), and deep learning improved first-order one-sided finite difference scheme (DFDM). As can be seen, DFDM has the smallest L_2 errors in all cases. Compared to the FDMs1, we obtain the largest improvement. The ratio denotes the error of the listed standard FDMs divided by the error of DFDM.

We illustrate the solution for two selected cases in Figure 3.8. As can be seen, FDMc and FDMs2 lead to spurious oscillations in the solution. FDMs1 does not cause any oscillations, but has a large error near the kink. DFDM produces the best solution among the methods.

We also computed the solution for the different final computation times T. The results are shown in Table 3.7. Let us note, that we only trained the method with the fixed $T = 1$, but we observe improving results also for different computation times. Based on the results presented, it can be stated that longer computational time leads to even better numerical results using DFDM.

parameters		L_2				improvement ratios		
σ	r	FDMs1	FDMs2	FDMc	DFDM	ratio (FDMs1)	ratio (FDMs2)	ratio (FDMc)
0.22	0.12	0.050304	0.016930	0.017986	**0.008781**	5.73	1.93	2.05
0.18	0.13	0.050193	0.021800	0.021481	**0.011393**	4.41	1.91	1.89
0.32	0.24	0.047173	0.010243	0.010361	**0.003628**	13.00	2.82	2.86
0.2	0.21	0.049032	0.018337	0.017864	**0.008794**	5.58	2.09	2.03
0.32	0.15	0.046425	0.010367	0.010167	**0.003600**	12.89	2.88	2.82
0.23	0.16	0.049239	0.015483	0.015921	**0.007031**	7.00	2.20	2.26
0.35	0.24	0.046521	0.009166	0.009298	**0.003086**	15.08	2.97	3.01
0.16	0.23	0.047185	0.025895	0.022651	**0.015085**	3.13	1.72	1.50
0.27	0.18	0.047725	0.012656	0.012479	**0.004896**	9.75	2.58	2.55
0.15	0.28	0.047074	0.029096	0.025772	**0.017967**	2.62	1.62	1.43

Table 3.6: Comparison of L_2 errors for the solution of the equation (3.29) with various parameters σ and r using different FDMs, $I = 50$, $T = 1$.



(a) $\sigma = 0.22$, $r = 0.12$.

(b) $\sigma = 0.16$, $r = 0.23$.

Figure 3.8: Comparison of the solution of the equation (3.29) using different FDMs, $I = 50$, $T = 1$.

parameters			L_2				improvement ratios		
σ	r	T	FDMs1	FDMs2	FDMc	DFDM	ratio (FDMs1)	ratio (FDMs2)	ratio (FDMc)
0.22	0.12	0.8	0.051032	0.019911	0.020737	**0.010952**	4.66	1.82	1.89
0.18	0.13	0.9	0.047113	0.023283	0.019891	**0.012144**	3.88	1.92	1.64
0.32	0.24	1.1	0.047800	0.009610	0.010600	**0.003927**	12.17	2.45	2.70
0.2	0.21	2	0.049840	0.011665	0.012847	**0.004838**	10.30	2.41	2.66
0.32	0.15	0.5	0.046685	0.016061	0.013966	**0.006675**	6.99	2.41	2.09
0.23	0.16	3	0.048125	0.007549	0.008789	**0.002720**	17.69	2.78	3.23
0.35	0.24	1.5	0.045988	0.006936	0.007465	**0.002200**	20.91	3.15	3.39
0.16	0.23	1.8	0.048583	0.017193	0.016256	**0.009683**	5.02	1.78	1.68
0.27	0.18	0.6	0.050456	0.018115	0.018972	**0.009517**	5.30	1.90	1.99
0.15	0.28	2.5	0.052187	0.014689	0.016639	**0.008149**	6.40	1.80	2.04

Table 3.7: Comparison of L_2 errors for the solution of the equation (3.29) with various parameters σ, r and T using different FDMs, $I = 50$.

Part II

Enhanced weighted essentially non-oscillatory schemes by deep learning

Chapter 4

4 Introduction to hyperbolic conservation laws and literature overview

4.1 Mathematical background for hyperbolic conservation laws

Typically, numerical fluid mechanics deals with nonlinear hyperbolic PDEs. In one space dimension, these equations can be represented as

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0, \qquad t > 0, \tag{4.1}$$

where x represents space, t denotes time, $u(x,t) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^m$ is an m-dimensional vector of conserved quantities and $f(u(x,t)) : \mathbb{R}^m \to \mathbb{R}^m$ is its flux. We refer to (4.1) as *hyperbolic conservation law* (HCL). That means, we assume that the $m \times m$ Jacobian matrix $f'(u)$ satisifies: For each value of u the eigenvalues of $f'(u)$ are real, and the matrix is diagonalizable [69].

Let us now introduce an integral form of the HCL (4.1) as

$$\begin{aligned}\int\limits_{x_1}^{x_2} u(x,t_2)dx = &\int\limits_{x_1}^{x_2} u(x,t_1)dx \\ &- \left[\int\limits_{t_1}^{t_2} f\big(u(x_2,t)\big)dt - \int\limits_{t_1}^{t_2} f\big(u(x_1,t)\big)dt\right]\end{aligned} \tag{4.2}$$

for x_1, x_2, t_1, t_2 from intervals $[x_1, x_2]$, $[t_1, t_2]$.

Applications

One of the most important systems of conservation laws is the Euler system of gas dynamics, which we also solve in this thesis (Chapter 6). As an introducing example we consider *the shock tube problem.* In this scenario, a tube is filled with gas, initially separated by a membrane into two sections. In one half of the tube, the gas exhibits higher density and pressure compared to the other half, while the velocity remains zero throughout. At the initial time, the membrane is suddenly removed, enabling the gas to flow. In this case the one-dimensional Euler equations (Section 6.1) are applicable.

Let us now describe three important waves that separate regions where the state variables are constant. A *shock wave* travels into the region of lower pressure, causing density and pressure to surge to higher levels, resulting in discontinuities in all state variables. This is followed by a *contact discontinuity*, characterised by another density discontinuity, while velocity and pressure remain constant. From the opposite direction comes the *rarefaction wave*, which causes all the state variables to remain continuous and a smooth transition to occur. This wave causes the density of the gas to decrease as it passes through.

Examples of HCLs include the further study of the propagation of these waves, as well as the investigation of conservation principles in various physical systems such as fluid dynamics, traffic flow and nonlinear acoustics.

Scalar conservation laws

Let us introduce the *linear advection equation* of a form

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0, \quad u(x,0) = u_0(x), \qquad t \geq 0, \tag{4.3}$$

with the solution $u(x,t) = u_0(x - at)$. This means, that the initial data is being transported unchanged to the left (if $a < 0$) or to the right (if $a > 0$) with the velocity a. It can be demonstrated that if $u_0(x)$ is a smooth function, $u_0 \in C^k(-\infty, \infty)$, then the solution $u(x,t)$ is also smooth both in space and time, $u \in C^k((-\infty,\infty) \times (0,\infty))$.

Along each ray $x - at = x_0$, which are known as the *characteristics* of the equation, the solution remains constant. Another important property of HCLs is that they have *finite propagation speed.* This means, that the characteristics have a finite slope. In other words, perturbations or changes in the system cannot spread instantaneously through space. This property is important for the design of numerical methods for HCLs.

Note that in the case of non-differentiability of u_0 at some point, $u(x,t)$ is no longer a classical solution. However, there exists a *weak solution*, provided that u_0 is integrable. For more details on weak solutions see [69].

Another important example of scalar conservation laws is the *Burgers' equation*

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \quad \text{with} \quad f(u) = \frac{u^2}{2}, \quad \text{i.e.} \quad f'(u) = u. \tag{4.4}$$

The characteristics are given by

$$x'(t) = u(x(t), t). \tag{4.5}$$

That is, they are straight lines, determined by the initial data. Assuming that the initial data is smooth, then this can be used to determine the solution $u(x, t)$ for sufficiently small t so that characteristics do not cross. However, the characteristics may intersect after some time, resulting in formation of a *shock*, where the discontinuity occurs and the function $u(x, t)$ has an infinite slope. Nevertheless, the weak solution exhibiting discontinuities still exists.

4.2 Numerical methods for hyperbolic conservation laws

It has been always been challenging to solve the HCLs numerically as it is well-known that discontinuities may develop after a finite time regardless of the smoothness of the initial or boundary data. Hence, suitable numerical methods must be designed for these problems, especially to approximate discontinuous solutions.

As the localisation of the shock is often the main cause of difficulties, so-called *shock-tracking* and *shock-capturing* schemes have been developed. The shock-tracking approach combines a conventional FDM in smooth regions with a special procedure designed to track the positions of discontinuities. However, this approach may become very complicated, especially in higher dimensional cases.

Therefore, so-called shock-capturing schemes became more popular. The numerical scheme is designed to detect and resolve shock waves implicitly as part of the solution process. Shock capturing methods aim to accurately capture the location, strength, and propagation of shocks by modifying the numerical scheme to prevent spurious oscillations and maintain stability near discontinuities.

To the beginning, let us consider the computational domain with a uniform grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0, \dots, I$. We also define the cell boundaries by $x_{i+\frac{1}{2}} = x_i + \frac{\Delta x}{2}$. The time domain will be discretized by the points $t_n = t_0 + n\Delta t$, $n = 0, \dots, N$.

We consider the advection equation (4.3) and we explain the construction of the numerical methods to approximate the solution $u_i^n = u(x_i, t_n)$. This approximation will be denoted by $\hat{u}_i^n$.

The most straightforward method to obtain the solution $\hat{u}_i^{n+1}$ would be to use centered finite difference approximation and explicit Euler time stepping such that it holds

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{2\Delta x} a(\hat{u}_{i+1}^n - \hat{u}_{i-1}^n). \tag{4.6}$$

However, this approach suffers from significant stability issues and is unlikely to be suitable for solving HCLs.

More suitable is so-called *upwinding approach.* For the scalar advection equation with $a > 0$, the one-sided method

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x} a(\hat{u}_i^n - \hat{u}_{i-1}^n), \tag{4.7}$$

is suitable. Accordingly, for $a < 0$ the upwind method has the form

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x} a(\hat{u}_{i+1}^n - \hat{u}_i^n). \tag{4.8}$$

Consistency, convergence and stability

Let us introduce a specific notation for the one-stage methods using

$$\hat{u}_i^{n+1} = \mathcal{H}_k(\hat{u}^n; i), \tag{4.9}$$

where we index $\mathcal{H}_k$ by a time step $k = \Delta t$. For example, considering equation (4.7) it holds

$$\mathcal{H}_k(\hat{u}^n; i) = \hat{u}_i^n - \frac{\Delta t}{\Delta x} a(\hat{u}_i^n - \hat{u}_{i-1}^n). \tag{4.10}$$

When this operator is applied to the piecewise constant function

$$\hat{u}_k(x,t) = \hat{u}_i^n, \quad \text{for} \quad (x,t) \in [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}) \times [t_n, t_{n+1}), \tag{4.11}$$

we obtain

$$\hat{u}_k(x, t+k) = \mathcal{H}_k(\hat{u}_k(\cdot, t); x). \tag{4.12}$$

Assuming that $\mathcal{H}_k$ is linear, we can write

$$\hat{u}^{n+1} = \mathcal{H}_k \hat{u}^n. \tag{4.13}$$

Let us now introduce the important metrics necessary to define convergence, consistency and stability. For classical solutions, the *global error* is defined by

$$E_i^n = \hat{u}_i^n - u_i^n. \tag{4.14}$$

Let us define the error function

$$E_k(x,t) = \hat{u}_k(x,t) - u(x,t), \tag{4.15}$$

using (4.11), which corresponds to the global error and

$$L_k(x,t) = \frac{1}{k}\big[u(x,t+k) - \mathcal{H}_k(u(\cdot,t); x)\big] \tag{4.16}$$

corresponding to the *local error.* Let us now define the basic properties of the numerical schemes, cf. [69].

Definition 2. *A method* $\hat{u}^{n+1} = \mathcal{H}_k(\hat{u}^n)$ *with the global error* (4.15) *is convergent with respect to a norm* $||\cdot||$, *if it holds*

$$\lim_{k\to 0} ||E_k(\cdot, t)|| = 0 \tag{4.17}$$

for any fixed $t \geq 0$ *and all initial data* u_0 *in some class.*

Definition 3. *A method* $\hat{u}^{n+1} = \mathcal{H}_k(\hat{u}^n)$ *with the local error* (4.16) *is called consistent, if it holds*

$$\lim_{k\to 0} ||L_k(\cdot, t)|| = 0 \tag{4.18}$$

for any fixed $t \geq 0$ *and all initial data* u_0 *in some class. The method is consistent of order* q, *if for each initial values with compact support, each* $T > 0$ *some constants* $C_L > 0$ *and* $k_0 > 0$ *exist such that*

$$||L_k(\cdot, t)|| \leq C_L k^q, \quad \text{for all} \quad k < k_0, t \leq T. \tag{4.19}$$

Definition 4. *A method* $\hat{u}^{n+1} = \mathcal{H}_k(\hat{u}^n)$ *is called stable (according to Lax-Richtmyer), if for each* $T \geq 0$ *some constants* $C_S > 0$ *and* $k_0 > 0$ *exist such that*

$$||\mathcal{H}_k^n|| \leq C_S \quad \text{for all} \quad k < k_0, nk \leq T. \tag{4.20}$$

According to the fundamental convergence theorem for linear FDMs it holds, that for a consistent, linear method, stability is necessary and sufficient for convergence.

4.2.1 Conservative methods

However, when we solve the nonlinear HCLs numerically (such as e.g. (4.4)), additional difficulties can arise. As mentioned in Section 4.1, discontinuous solutions and shocks are often present and the numerical method may converge to the wrong solution. Therefore, the *conservative methods* are required, which are in the form [69]

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x}[\hat{f}(\hat{u}_{i-p}^n, \hat{u}_{i-p+1}^n, \dots \hat{u}_{i+s}^n) - \hat{f}(\hat{u}_{i-p-1}^n, \hat{u}_{i-p}^n, \dots \hat{u}_{i+s-1}^n)]. \tag{4.21}$$

where $\hat{f}$ is the *numerical flux function* of $p+s+1$ arguments. To obtain a reasonable approximation, a natural requirement is that a constant flux function is exactly approximated, so that

$$\hat{f}(u, u, \dots, u) = f(u). \tag{4.22}$$

In the simplest case, taking $p = 0$ and $s = 1$ we obtain

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x}\Big[\hat{f}(\hat{u}_i^n, \hat{u}_{i+1}^n) - \hat{f}(\hat{u}_{i-1}^n, \hat{u}_i^n)\Big]. \tag{4.23}$$

According to the equation (4.2), integral form of the HCL (4.1) yields the equation

$$\begin{aligned}\int\limits_{x_{i-1/2}}^{x_{i+1/2}} u(x, t_{n+1})dx = &\int\limits_{x_{i-1/2}}^{x_{i+1/2}} u(x, t_n)dx \\ &- \Bigg[\int\limits_{t_n}^{t_{n+1}} f\big(u(x_{i+1/2}, t)\big)dt - \int\limits_{t_n}^{t_{n+1}} f\big(u(x_{i-1/2}, t)\big)dt\Bigg].\end{aligned} \tag{4.24}$$

Using shorter notation for (4.21) we can write

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x}\Big[\hat{f}(\hat{u}^n; i) - \hat{f}(\hat{u}^n; i-1)\Big] \tag{4.25}$$

and we can see $\hat{f}(\hat{u}^n; i)$ as approximation of the average flux

$$\hat{f}(\hat{u}^n; i) \approx \frac{1}{\Delta t}\int\limits_{t_n}^{t_{n+1}} f\big(u(x_{i+1/2}, t)\big)dt. \tag{4.26}$$

For example, considering the Burgers' equation (4.4) and the upwind scheme (4.7), we would obtain

$$\hat{u}_i^{n+1} = \hat{u}_i^n - \frac{\Delta t}{\Delta x}\Big(\frac{1}{2}(\hat{u}_i^n)^2 - \frac{1}{2}(\hat{u}_{i-1}^n)^2\Big), \tag{4.27}$$

which has the conservative form (4.23) with

$$\hat{f}(v, w) = f(v) = \frac{1}{2}v^2. \tag{4.28}$$

Definition 5. *A conservative method* (4.21) *is called consistent, if the local Lipschitz condition*

$$|\hat{f}(\hat{u}_{i-p}, \dots \hat{u}_{i+s}) - f(u)| \leq K \max_{-p \leq j \leq s} |\hat{u}_{i+j} - u| \tag{4.29}$$

holds for all $\hat{u}_{i+j}$ *sufficiently close to* u *with a constant* K*.*

Note, that (4.29) implies (4.22).

We note that according to the Lax-Wendroff theorem [66], if the solution of the conservative scheme (4.25) converges, it will converge to a weak solution of (4.1).

Total variation diminishing methods

We will further apply the *total variation diminishing* (TVD) time-stepping method in this thesis, so let us briefly introduce the following terms, cf. [69].

Definition 6. *Let $T > 0$ be a given constant. For a given function $u = u(x,t)$ the total variation over $[0,T]$ is defined by*

$$TV(u) = \sup \sum_{i=1}^{I} \big| u(x_i) - u(x_{i-1}) \big|, \tag{4.30}$$

where the supremum is taken over all subdivisions of the real line $-\infty = x_0 < x_1 < \ldots < x_I = \infty$.

Definition 7. *The numerical method $\hat{u}_i^{n+1} = \mathcal{H}(\hat{u}^n; i)$ is called total variation diminishing if*

$$TV(\hat{u}^{n+1}) \leq TV(\hat{u}^n) \tag{4.31}$$

for all grid functions $\hat{u}^n$.

4.3 Related scientific work

Conservative numerical methods are constructed using relations listed in Section 4.2.1. The *Lax-Wendroff method* [66] and the *Godunov method* [32] were among the earliest FDMs developed for solving hyperbolic conservation laws. Later, so-called *flux-limiter* methods have been introduced, cf. [110]. The main idea of this flux splitting approach is to decompose the flux by choosing a high order flux suitable for smooth regions, and a low order flux that works well near discontinuities. Also very important are the *TVD methods* [38], which were developed to maintain or reduce the total variation of the solution over time, ensuring stability and accuracy near shocks and discontinuities.

In 1980 Crandall and Majda [19] proposed the class of *monotone schemes* which are nonlinearly stable in the L_1 norm and satisfy certain entropy conditions. It can be shown that the corresponding solutions converge to bounded variation entropy solutions including error estimates. However, these schemes are only first-order accurate, and it is known from the fundamental Godunov theorem [31] that one must consider nonlinear non-oscillatory schemes to overcome this accuracy limit.

In this direction shock-capturing schemes have been developed which are able to resolve a shock or a steep gradient region sharply without introducing too much diffusion or overshoot behaviour [38]. The well-known representative of this class of methods are the *essentially non-oscillatory* (ENO) schemes [39] with high order accuracy in smooth regions and sharply resolving shocks in an essentially non-oscillatory manner using a *smoothness indicator function*, see e.g. [98]. Later, Jiang and Shu [50] further improved these schemes and proposed a *weighted essentially non-oscillatory scheme* (subsequently abbreviated as WENO-JS), which is still considered a state-of-the-art solution.

As we continue to develop the WENO schemes in this thesis, we present the development of these methods in more detail. In order to achieve higher order accuracy for WENO schemes, the stencil for the spatial reconstruction needs to be enlarged and to keep it in a compact size Qiu and Shu [82, 83] developed the *hermite WENO* (HWENO) schemes. To further increase the efficiency, Pirozzoli [81] developed a *hybrid compact-WENO scheme* using upwind schemes in the smooth regions. Alternatively, Hill and Pullin [43] designed another hybrid WENO scheme, combining special centred difference schemes with WENO methods, cf. [119] for the most recent approaches.

Subsequently, several new strategies were developed by modifying the WENO-JS schemes, i.e. by modifying the nonlinear weights [13, 15, 34, 42, 54, 71, 88]. In addition, another goal in optimizing these schemes has been to minimize the dispersion error (dispersion-relation-preserving (DRP) schemes) [73, 107], also combined with the WENO approach leading to *optimized WENO* (OWENO) schemes [114]. Since classical WENO methods are too dissipative for *direct numerical simulations* (DNS) of turbulence, the aim has been to reduce the dissipation by incorporating an automatic dissipation adjustment [27].

In 2016 Fu et al. [29] proposed a family of high-order *targeted ENO* (TENO) schemes that are particularly suitable for DNS. These methods reduce the numerical dissipation by an ENO-like stencil selection and increase the numerical robustness by assembling a set of low-order candidate stencils with increasing width, see also the extensions in [28, 30]. For a detailed review of WENO schemes we refer the interested reader to [99].

Besides HCLs, the WENO schemes for nonlinear degenerate parabolic equations have also been developed. The behaviour of these equations is very similar to that of HCLs. Therefore, the well-known WENO method, which is widely used for solving HCLs, has also been generalized for these equations.

For this purpose, the authors Liu et al. developed the WENO scheme for nonlinear degenerate parabolic equations [74]. Two formulations are described in [74]. In the first, the second derivative is directly approximated by a conservative flux difference. In this case, the negative ideal weights appear, so a special treatment of these weights is required [97]. The desired sixth-order accuracy is achieved and numerically demonstrated. The second approach is based on introducing an auxiliary variable for the first derivative, then applying the WENO scheme to two first derivatives instead of the second derivative. However, this case is not discussed further because the magnitude of the error is greater than when the WENO method is applied directly to the second derivative.

Subsequently, new modifications of the sixth-order WENO method for nonlinear degenerate parabolic equations were introduced. Christlieb et al. [18] provided a high-order WENO method with a nonlinear filter to avoid unwanted spurious oscillations. Hajipour and Malek [35] introduced a new type of nonlinear weights and used a non-standard Runge-Kutta scheme instead of the TVD Runge-Kutta scheme [100] previously used in combination with WENO methods. Abedian et al. [1, 2] aimed to avoid the negative ideal weights and present a new modification of the WENO method. Rathan et al. [87] developed a new smoothness indicator based on the L_1 norm. Jiang [51] developed another WENO method for nonlinear degenerate parabolic equations.

Recently, machine learning has also been used to improve standard WENO schemes. In [113], deep reinforcement learning is used to design a new numerical scheme for solving HCLs. The authors apply their method to the solution of Burgers' equation and compare their results with the standard WENO scheme. The recent work of Stevens and Colonius [105] introduces a new WENO-NN scheme based on a NN algorithm. In their work, the finite volume coefficients of the WENO-JS scheme are perturbed while retaining the original smoothness indicators and nonlinear weights. However, the resulting scheme presented in their paper has only first order accuracy. Another NN based WENO scheme was developed by Liu and Wen [72], where the new smoothness indicators are an output of the NN algorithm. However, in this case, the formal order of accuracy of the reconstruction of the resulting method can neither be analytically proven nor guaranteed. For a detailed discussion of the formal order of accuracy (as opposed to the term "convergence") for WENO methods, see e.g. [4, 21].

In our work, we implement another WENO extension based on deep learning. This approach is used to improve the classic WENO-JS [50] and WENO-Z [13] schemes in this work, but could also be efficiently applied to other WENO methods. For this purpose, we will train a rather small NN to perturb the smoothness indicator functions of the WENO-JS scheme. Since we do not develop new smoothness indicators as in [72], but only their multiplicative perturbations, we are able to prove the formal order of accuracy of the resulting scheme. We call this new scheme WENO-DS (Deep Smoothness) because we modify the smoothness indicators using deep NNs. This scheme has less diffusion and less overshoot in shocks than the WENO-JS and WENO-Z schemes, while maintaining high order accuracy in smooth regions. Furthermore, we efficiently implement the WENO-DS scheme for solving one- and two-dimensional Euler equations of gas dynamics.

We generalize this algorithm also for nonlinear degenerate parabolic equations. We use a NN algorithm to modify the smoothness indicators of the original WENO scheme [35, 74] and obtain sixth-order accuracy, which we prove theoretically. We emphasize that no post-processing steps are required to maintain the consistency and accuracy of the method.

Finally, we apply the WENO-DS method to a computational finance problem, namely the European digital option pricing problem with discontinuous terminal data. In this problem, the spurious oscillations are present in the solution when the standard WENO scheme is used. We show that they can be successfully eliminated using the WENO-DS method.

5

Chapter 5

Fifth-order WENO scheme for hyperbolic conservation laws

In this chapter we enhance the well-known fifth-order WENO shock-capturing scheme by using deep learning techniques. This fine-tuning of an existing algorithm is implemented by training a rather small NN to modify the smoothness indicators of the WENO scheme in order to improve the numerical results especially at discontinuities. In our approach no further post-processing is needed to ensure the consistency of the method. Moreover, the formal order of accuracy of the resulting scheme can be theoretically proven.

We demonstrate our findings with the Buckley-Leverett equation and inviscid Burgers' equation. We show that our novel method outperforms the classical fifth-order WENO schemes in simulations where the numerical solution is too diffusive or tends to overshoot at shocks. Finally, the straight-forward extension of the method to two-dimensional problems is included and illustrated using the two-dimensional Burgers' equation. Further in Chapter 6 we present the application to the one-dimensional and two-dimensional Euler equations of gas dynamics.

5.1 The WENO scheme

Let us consider the one-dimensional HCL of a form

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0, \qquad t > 0. \tag{5.1}$$

We introduce a uniform grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0, \dots, I$, which are the centers of the cells with cell boundaries defined by $x_{i+\frac{1}{2}} = x_i + \frac{\Delta x}{2}$.

The spatial discretization of one-dimensional HCL (5.1) yields a system of ODEs and the resulting semi-discrete scheme is

$$\frac{du_i(t)}{dt} = -\frac{\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}}{\Delta x}, \tag{5.2}$$

where $u_i(t)$ approximates pointwise $u(x_i, t)$ and the numerical flux $\hat{f}_{i+\frac{1}{2}}$ is chosen such that for all sufficiently smooth u, we have

$$\frac{1}{\Delta x}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) = \big(f(u)\big)_x|_{x=x_i} + O(\Delta x^5), \tag{5.3}$$

with fifth-order of accuracy. Recalling the equation (4.21), we see, that the form (5.2) corresponds to this conservative scheme with a numerical flux function

$$\hat{f}_{i+\frac{1}{2}} = \hat{f}(u_{i-p}, \dots u_{i+s}), \tag{5.4}$$

which satisfies the following conditions: $\hat{f}$ is a Lipschitz continuous function in all the arguments and $\hat{f}$ is consistent with the physical flux f, that means, $\hat{f}(u, \dots, u) = f(u)$. By u_i we denote the pointwise values $u_i = u(x_i)$. To satisfy (5.3), $p = 2$ and $s = 2$ are chosen [98].

Following [50], if we define a function h implicitly by

$$f\big(u(x)\big) = \frac{1}{\Delta x}\int\limits_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} h(\xi)\, d\xi, \tag{5.5}$$

then (5.2) is approximated by

$$\big(f(u)\big)_x|_{x=x_i} = \frac{h\big(x + \frac{\Delta x}{2}\big) - h\big(x - \frac{\Delta x}{2}\big)}{\Delta x}. \tag{5.6}$$

Considering $h_{i\pm\frac{1}{2}} = h(x_{i\pm\frac{1}{2}})$ we obtain the following fifth-order approximation of a numerical flux

$$\hat{f}_{i\pm\frac{1}{2}} = h_{i\pm\frac{1}{2}} + O(\Delta x^5). \tag{5.7}$$

This results in a conservative numerical scheme.

Further, it is essential to use the *flux splitting method*, thus we write

$$f(u) = f^+(u) + f^-(u), \quad \text{where} \quad \frac{df^+(u)}{du} \geq 0 \quad \text{and} \quad \frac{df^-(u)}{du} \leq 0. \tag{5.8}$$

The numerical flux $\hat{f}_{i\pm\frac{1}{2}}$ is then represented by $\hat{f}_{i\pm\frac{1}{2}} = \hat{f}^+_{i\pm\frac{1}{2}} + \hat{f}^-_{i\pm\frac{1}{2}}$ and the final scheme is formed as

$$\frac{du_i(t)}{dt} = -\frac{1}{\Delta x}\left[\left(\hat{f}^+_{i+\frac{1}{2}} - \hat{f}^+_{i-\frac{1}{2}}\right) + \left(\hat{f}^-_{i+\frac{1}{2}} - \hat{f}^-_{i-\frac{1}{2}}\right)\right]. \tag{5.9}$$

Next, we only consider the construction of $\hat{f}^+_{i\pm\frac{1}{2}}$ (and drop the superscript +). The negative part can be then obtained using symmetry (see e.g. [112]).

5.1.1 Fifth-order WENO scheme

For a construction of $\hat{f}_{i+\frac{1}{2}}$, the fifth-order WENO method uses a 5-point stencil

$$S(i) = \{x_{i-2}, \dots, x_{i+2}\} \tag{5.10}$$

divided into three candidate substencils, which are given by

$$S(i)^m = \{x_{i+m-2}, x_{i+m-1}, x_{i+m}\}, \quad m = 0, 1, 2. \tag{5.11}$$

To form the numerical flux over the entire 5-point stencil, the numerical flux for each of these substencils $\hat{f}^m_{i+\frac{1}{2}} = h_{i+\frac{1}{2}} + O(\Delta x^3)$ is calculated. These fluxes are then averaged in such a way, that fifth-order accuracy is ensured in the smooth regions. In regions with discontinuities, the weights should partly remove the contribution of these stencils so that the solution near the shock can be approximated in more stable manner.

Let $\hat{f}^m(x)$ be the polynomial approximation of $h(x)$ on each of the substencils (5.11). Then, evaluated at $x = x_{i+\frac{1}{2}}$ we obtain

$$\hat{f}^m(x_{i+\frac{1}{2}}) = \hat{f}^m_{i+\frac{1}{2}} = \sum_{j=0}^{2} c_{m,j}\, f(u_{i+m-2+j}), \quad i = 0, \dots, I, \tag{5.12}$$

where $c_{m,j}$ are the Lagrangian interpolation coefficients, dependent on m (see [50]). They take an explicit form

$$\begin{aligned} \hat{f}^0_{i+\frac{1}{2}} &= \frac{2f(u_{i-2}) - 7f(u_{i-1}) + 11f(u_i)}{6}, \\ \hat{f}^1_{i+\frac{1}{2}} &= \frac{-f(u_{i-1}) + 5f(u_i) + 2f(u_{i+1})}{6}, \\ \hat{f}^2_{i+\frac{1}{2}} &= \frac{2f(u_i) + 5f(u_{i+1}) - f(u_{i+2})}{6}, \end{aligned} \tag{5.13}$$

and the numerical fluxes $\hat{f}^m_{i-\frac{1}{2}}$ can be obtained by shifting each index by -1. By $f(u_i)$ the value of a function f at $u(x_i)$ is indicated.

Using the Taylor series expansion it can be shown that:

$$\begin{aligned} \hat{f}^0_{i\pm\frac{1}{2}} &= h_{i\pm\frac{1}{2}} - \frac{1}{4}\frac{d^3 f}{dx^3}\Big|_{x=x_i} \Delta x^3 + O(\Delta x^4), \\ \hat{f}^1_{i\pm\frac{1}{2}} &= h_{i\pm\frac{1}{2}} + \frac{1}{12}\frac{d^3 f}{dx^3}\Big|_{x=x_i} \Delta x^3 + O(\Delta x^4), \\ \hat{f}^2_{i\pm\frac{1}{2}} &= h_{i\pm\frac{1}{2}} - \frac{1}{12}\frac{d^3 f}{dx^3}\Big|_{x=x_i} \Delta x^3 + O(\Delta x^4). \end{aligned} \tag{5.14}$$

Doing so, we obtain the general form of these expressions

$$\hat{f}^m_{i\pm\frac{1}{2}} = h_{i\pm\frac{1}{2}} + A_m \Delta x^3 + O(\Delta x^4), \tag{5.15}$$

with A_m being independent of Δx.

Then, the convex combination of the interpolated values $\hat{f}^m_{i\pm\frac{1}{2}}$ given by

$$\hat{f}_{i\pm\frac{1}{2}} = \sum_{j=0}^{2} \omega_m \, \hat{f}^m_{i\pm\frac{1}{2}} \tag{5.16}$$

yields the WENO approximation of the value $h_{i\pm\frac{1}{2}}$, where ω_m are the nonlinear weights defined as, cf. [50]

$$\omega_m^{JS} = \frac{\alpha_m^{JS}}{\sum_{i=0}^{2} \alpha_i^{JS}}, \quad \text{where} \quad \alpha_m^{JS} = \frac{d_m}{(\epsilon + \beta_m)^2}. \tag{5.17}$$

The scheme using these nonlinear weights is denoted as the WENO-JS scheme. The parameter ϵ guarantees that the denominator does not become zero and should be chosen carefully, as it can change the order of accuracy of the scheme [4, 42]. The coefficients d_0, d_1 and d_2 are called ideal weights, which would form the upstream fifth-order central scheme for the 5-point stencil and satisfy (5.7). Their values are:

$$d_0 = \frac{1}{10}, \quad d_1 = \frac{6}{10}, \quad d_2 = \frac{3}{10}. \tag{5.18}$$

The parameter β_m is called the *smoothness indicator* and is analyzed in Section 5.1.2.

5.1.2 Smoothness indicators

The role of smoothness indicators is to measure the regularity of the polynomial approximation $\hat{f}^m(x)$ in each of three substencils (5.11). As developed in [50], they are defined as:

$$\beta_m = \sum_{q=1}^{2} \Delta x^{2q-1} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left(\frac{d^q \hat{f}^m(x)}{dx^q} \right)^2 dx. \tag{5.19}$$

Corresponding to the flux approximation $\hat{f}_{i+\frac{1}{2}}$ they take an explicit form

$$\begin{aligned}
\beta_0 &= \frac{13}{12}\big(f(u_{i-2}) - 2f(u_{i-1}) + f(u_i)\big)^2 + \frac{1}{4}\big(f(u_{i-2}) - 4f(u_{i-1}) + 3f(u_i)\big)^2,\\
\beta_1 &= \frac{13}{12}\big(f(u_{i-1}) - 2f(u_i) + f(u_{i+1})\big)^2 + \frac{1}{4}\big(-f(u_{i-1}) + f(u_{i+1})\big)^2,\\
\beta_2 &= \frac{13}{12}\big(f(u_i) - 2f(u_{i+1}) + f(u_{i+2})\big)^2 + \frac{1}{4}\big(3f(u_i) - 4f(u_{i+1}) + f(u_{i+2})\big)^2,
\end{aligned} \tag{5.20}$$

and their Taylor expansions at x_i are:

$$\begin{aligned}
\beta_0 &= f_x^2\Delta x^2 + \Big(\frac{13}{12}f_{xx}^2 - \frac{2}{3}f_x f_{xxx}\Big)\Delta x^4 \\
&\quad + \Big(-\frac{13}{6}f_{xx}f_{xxx} + \frac{1}{2}f_x f_{xxxx}\Big)\Delta x^5 + O(\Delta x^6), \\
\beta_1 &= f_x^2\Delta x^2 + \Big(\frac{13}{12}f_{xx}^2 + \frac{1}{3}f_x f_{xxx}\Big)\Delta x^4 + O(\Delta x^6), \qquad (5.21) \\
\beta_2 &= f_x^2\Delta x^2 + \Big(\frac{13}{12}f_{xx}^2 - \frac{2}{3}f_x f_{xxx}\Big)\Delta x^4 \\
&\quad + \Big(\frac{13}{6}f_{xx}f_{xxx} - \frac{1}{2}f_x f_{xxxx}\Big)\Delta x^5 + O(\Delta x^6),
\end{aligned}$$

where we used the short notation for the derivatives $f_x = f(u_i)_x$. These indicators are designed to come closer to zero for smooth parts of the solution so that the nonlinear weights ω_m come closer to the ideal weights d_m. In the case that the stencil S^m contains a discontinuity, β_m is $O(1)$ and the corresponding weight ω_m becomes smaller, therefore the contribution of the substencil S^m is reduced.

Following the work of Henrick, Aslam and Powers [42], it can be shown that demanding (5.7) we obtain the sufficient conditions for the fifth-order accuracy:

$$\sum_{m=0}^{2}(\omega_m^{\pm} - d_m) = O(\Delta x^6), \qquad (5.22)$$

$$\omega_m^{\pm} - d_m = O(\Delta x^3). \qquad (5.23)$$

Considering the overall finite difference formula

$$\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}} = f'(x)\Delta x + O(\Delta x^6),$$

it can be shown, that (5.23) may be relaxed and we obtain the following sufficient and necessary conditions:

$$\sum_{m=0}^{2}(\omega_m^{\pm} - d_m) = O(\Delta x^6), \qquad (5.24)$$

$$\sum_{m=0}^{2}A_m(\omega_m^{+} - \omega_m^{-}) = O(\Delta x^3), \qquad (5.25)$$

$$\omega_m^{\pm} - d_m = O(\Delta x^2). \qquad (5.26)$$

Note that due to the normalization (5.17), the first condition (5.24) (resp. (5.22)) is always fulfilled. The superscripts ± on ω_m specify their use in $\hat{f}_{i+\frac{1}{2}}$ or $\hat{f}_{i-\frac{1}{2}}$.

The analysis of the formal order of accuracy was performed in [50] and it was shown that if

$$\beta_m = D\big(1 + O(\Delta x^2)\big), \qquad (5.27)$$

with D being a non-zero constant independent of m, the condition (5.26) is satisfied.

However, it was shown in [42] that at the critical points where the first derivative of f vanishes, the order of accuracy of the scheme from [50] decreases to the third order. Moreover, if the second derivative also vanishes, the accuracy order is further reduced to the second order. For a further explanation of this problem we refer the interested reader to [42].

Remark 2. As mentioned before, we only considered the construction of the numerical flux $\hat{f}^+_{i+\frac{1}{2}}$. For the numerical approximation of the flux $\hat{f}^+_{i-\frac{1}{2}}$ we can use formulas (5.13), (5.16), (5.17) and (5.20) and shift each index by -1.

Remark 3. The negative part of the flux splitting can be obtained using symmetry (see, e.g., [112]), and we briefly summarize the formulas for $\hat{f}^-_{i+\frac{1}{2}}$ and omit the superscript $^-$:

$$\begin{aligned}
\hat{f}^0_{i+\frac{1}{2}} &= \frac{11f(u_{i+1}) - 7f(u_{i+2}) + 2f(u_{i+3})}{6},\\
\hat{f}^1_{i+\frac{1}{2}} &= \frac{2f(u_i) + 5f(u_{i+1}) - f(u_{i+2})}{6},\\
\hat{f}^2_{i+\frac{1}{2}} &= \frac{-f(u_{i-1}) + 5f(u_i) + 2f(u_{i+1})}{6},
\end{aligned} \tag{5.28}$$

where the weights ω^{JS}_m are computed as in (5.17) using the smoothness indicators given by

$$\begin{aligned}
\beta_0 &= \frac{13}{12}\big(f(u_{i+1}) - 2f(u_{i+2}) + f(u_{i+3})\big)^2 + \frac{1}{4}\big(3f(u_{i+1}) - 4f(u_{i+2}) + f(u_{i+3})\big)^2,\\
\beta_1 &= \frac{13}{12}\big(f(u_i) - 2f(u_{i+1}) + f(u_{i+2})\big)^2 + \frac{1}{4}\big(f(u_i) - f(u_{i+2})\big)^2,\\
\beta_2 &= \frac{13}{12}\big(f(u_{i-1}) - 2f(u_i) + f(u_{i+1})\big)^2 + \frac{1}{4}\big(f(u_{i-1}) - 4f(u_i) + 3f(u_{i+1})\big)^2.
\end{aligned} \tag{5.29}$$

In Section 5.2, where the deep learning algorithm will be introduced, this will help to understand how the improved smoothness indicators will be constructed.

5.1.3 The WENO-Z scheme

As it was mentioned in [42] or [13], the original WENO-JS looses its fifth-order of accuracy in the critical points, where the first derivative of f vanishes. Therefore, a new modification of the scheme was developed by Borges et al. [13] and we introduce it in this section. A new global smoothness indicator is characterized by

$$\tau_5 = |\beta_0 - \beta_2|. \tag{5.30}$$

It is easy to see from the equations (5.21) that

$$\tau_5 = \left| -\frac{13}{3} f_{xx} f_{xxx} + f_x f_{xxxx} \right| \Delta x^5 + O(\Delta x^6). \tag{5.31}$$

The new WENO-Z weights are then defined by

$$\omega_m^Z = \frac{\alpha_m^Z}{\sum_{i=0}^{2} \alpha_i^Z}, \qquad \text{where} \qquad \alpha_m^Z = d_m \left[1 + \left(\frac{\tau_5}{\beta_m + \epsilon} \right)^2 \right]. \tag{5.32}$$

Borges et al. [13] have shown that when using these nonlinear weights, fifth-order accuracy is preserved, even at the critical points where $f'(u) = 0$. Besides the improvement concerning accuracy, WENO-Z is also considered to be less dissipative than WENO-JS scheme and producing sharp and more accurate numerical solutions. For our implementation, we use WENO-Z with its weights as base method, which we will improve by enhancing its smoothness indicators. Let us note, that our approach could be generalized for another families of WENO schemes as well.

5.2 Application of deep learning to the fifth-order WENO scheme

To better capture discontinuities and avoid oscillations, we propose to apply deep learning to develop new smoothness indicators. We construct them as products of the original smoothness indicators β_m and multipliers δ_m which are outputs of a neural network algorithm. We refer to these new smoothness indicators as β_m^{DS}, where index DS corresponds to the new WENO-DS scheme:

$$\beta_m^{DS} = \beta_m(\delta_m + C), \tag{5.33}$$

where C is a constant, whose role is crucial for the proof of consistency and the formal order of accuracy and we will explain how to choose it in Section 5.2.1. We point out that this formulation as a multiplication is very advantageous in a sense that the consistency and accuracy properties can be analytically shown. In the case that the solution is smooth and the original smoothness indicator β_m converges to zero, the smoothness indicator β_m^{DS} behaves in the same way. If the smoothness indicator β_m is $O(1)$, the multiplier δ_m can change it so that the final scheme performs better. We emphasize, that there was an attempt by Liu and Wen [72] to learn the smoothness indicators directly. However, in this case the consistency and accuracy analysis could not be performed.

According to equation (5.9) we have to consider the new smoothness indicators for the positive part $\hat{f}^+_{i\pm\frac{1}{2}}$ of a numerical flux, as well as for the negative part $\hat{f}^-_{i\pm\frac{1}{2}}$ of a numerical flux. For the positive part $\hat{f}^+_{i\pm\frac{1}{2}}$ we define the new smoothness indicators:

$$\begin{aligned} \beta^{DS}_{m,i+\frac{1}{2}} &= \beta_{m,i+\frac{1}{2}}(\delta_{m,i+\frac{1}{2}} + C), \\ \beta^{DS}_{m,i-\frac{1}{2}} &= \beta_{m,i-\frac{1}{2}}(\delta_{m,i-\frac{1}{2}} + C), \end{aligned} \tag{5.34}$$

with

$$\delta_{0,i+\frac{3}{2}} = \delta_{1,i+\frac{1}{2}} = \delta_{2,i-\frac{1}{2}}, \quad i = 0, \dots, I. \tag{5.35}$$

Here, $\beta^{DS}_{m,i+\frac{1}{2}}$, $m = 0, 1, 2$ represent the smoothness indicators corresponding to the numerical fluxes $\hat{f}^m_{i+\frac{1}{2}}$, $m = 0, 1, 2$. The smoothness indicators $\beta^{DS}_{m,i-\frac{1}{2}}$, $m = 0, 1, 2$ correspond to the numerical fluxes $\hat{f}^m_{i-\frac{1}{2}}$, $m = 0, 1, 2$. Due to the definition, the values $\beta^{DS}_{m,i-\frac{1}{2}}$ can be obtained from $\beta^{DS}_{m,i+\frac{1}{2}}$ by simple index shift, which further holds also for obtaining the values $\hat{f}^m_{i-\frac{1}{2}}$ from $\hat{f}^m_{i+\frac{1}{2}}$ and the conservative property is preserved.

For the negative part $\hat{f}^-_{i\pm\frac{1}{2}}$ it holds analogously:

$$\begin{aligned} \beta^{DS}_{m,i+\frac{1}{2}} &= \beta_{m,i+\frac{1}{2}}(\delta_{m,i+\frac{1}{2}} + C), \\ \beta^{DS}_{m,i-\frac{1}{2}} &= \beta_{m,i-\frac{1}{2}}(\delta_{m,i-\frac{1}{2}} + C), \end{aligned} \tag{5.36}$$

with

$$\delta_{0,i-\frac{1}{2}} = \delta_{1,i+\frac{1}{2}} = \delta_{2,i+\frac{3}{2}}, \quad i = 0, \dots, I. \tag{5.37}$$

In Section 5.3 we will present whole algorithm of the method as well as an illustrative image, which explains, from which values the multipliers will be constructed.

5.2.1 Accuracy analysis

In this section, we perform the accuracy analysis of the WENO-DS scheme. We show, that using new smoothness indicators in the original WENO-JS scheme we can not guarantee the formal fifth-order of accuracy. However, we prove the formal fifth-order of accuracy when we use the smoothness indicators $\beta^{DS}_{m,i\pm\frac{1}{2}}$, $m = 0, 1, 2$ for the nonlinear weights of the original WENO-Z scheme described in Section 5.1.3.

We perform the analysis for the positive part $\hat{f}^+_{i\pm\frac{1}{2}}$ of the flux, as for the negative part it can be done analogously.

Initially, we establish an assumption that will be subsequently required.

Assumption 5.2.1. *Let us assume, that in each time step there exists a function* $\Phi \in \mathbb{R}^{2k+1} \to \mathbb{R}$*, such that the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *from* (5.34) *in the node* x_i *satisfying*

(5.35) *can be expressed as:*

$$\begin{aligned}\delta_{0,i+\frac{1}{2}} &= \Phi(\bar{x}_i - \Delta x) = \Phi(\bar{x}_i) - O(\Delta x),\\ \delta_{1,i+\frac{1}{2}} &= \Phi(\bar{x}_i),\\ \delta_{2,i+\frac{1}{2}} &= \Phi(\bar{x}_i + \Delta x) = \Phi(\bar{x}_i) + O(\Delta x),\end{aligned} \tag{5.38}$$

where

$$\bar{x}_i = (x_{i-k}, x_{i-k+1}, \dots, x_{i+k}). \tag{5.39}$$

Moreover, let us assume that there exists a constant C independent from time step and discretization, such that it holds $\Phi(\bar{x}_i) + C > \kappa > 0$ with κ fixed, $i = 0, \dots, I$.

Accuracy analysis of WENO-JS scheme with smoothness indicators β_m^{DS}

Using (5.27) and with the Assumption 5.2.1, it holds

$$\beta^{DS}_{m,i\pm\frac{1}{2}} = \beta_{m,i\pm\frac{1}{2}}(\delta_{m,i\pm\frac{1}{2}} + C) = D\big(1 + O(\Delta x^2)\big)\big(\Phi(\bar{x}_i) + O(\Delta x) + C\big), \tag{5.40}$$

with D being some non-zero constant independent of m. We denote $P(\bar{x}_i) = \Phi(\bar{x}_i) + C$ and we set C such that it satisfies the Assumption 5.2.1. Then we ensure that $P(\bar{x}_i) = O(1)$. Performing the multiplication in (5.40) we obtain

$$\begin{aligned}\beta^{DS}_{m,i\pm\frac{1}{2}} &= D\big(P(\bar{x}_i) + P(\bar{x}_i)O(\Delta x^2) + O(\Delta x) + O(\Delta x^3)\big)\\ &= DP(\bar{x}_i)\big(1 + O(\Delta x)\big) = \tilde{D}\big(1 + O(\Delta x)\big).\end{aligned} \tag{5.41}$$

Here we can proceed as in [42], but for the reader's convenience we repeat the steps of the proof: insert (5.41) into (5.17) and take $\epsilon = 0$

$$\alpha^{DS}_{m,i\pm\frac{1}{2}} = \frac{d_m}{\big(\tilde{D}(1 + O(\Delta x))\big)^2} = \frac{d_m}{\tilde{D}^2}\big(1 + O(\Delta x)\big). \tag{5.42}$$

This implies that

$$\sum_{m=0}^{2} \alpha^{DS}_{m,i\pm\frac{1}{2}} = \frac{1}{\tilde{D}^2}\big(1 + O(\Delta x)\big), \tag{5.43}$$

where we used the fact that $\sum_{m=0}^{2} d_m = 1$. Finally, substituting into (5.17) we obtain

$$\omega^{DS}_{m,i\pm\frac{1}{2}} = d_m + O(\Delta x), \tag{5.44}$$

where the superscript DS denotes the enhancement of the nonlinear weights (5.17) using our novel method. We see, that neither the condition (5.23), nor (5.26) is satisfied. However, as (5.44) holds, we can still guarantee that for the WENO-JS scheme with the smoothness indicators (5.34) we have a formal order of accuracy degraded to the third order, cf. Borges et al. [13].

Accuracy analysis of WENO-Z scheme with smoothness indicators β_m^{DS}

Here we show, that inserting the nonlinear weights ω_m^{DS}, $m = 0, 1, 2$ into the original WENO-Z scheme [13] we ensure the formal fifth-order of accuracy, which can be theoretically proven.

Theorem 1. *Let the numerical flux of the WENO-DS scheme be given by* (5.13) *and* (5.16) *with the corresponding nonlinear weights given by*

$$\omega_m^{DS} = \frac{\alpha_m^{DS}}{\sum_{i=0}^{2} \alpha_i^{DS}}, \qquad \alpha_m^{DS} = d_m \left[1 + \left(\frac{\tau_5}{\beta_{m,i\pm\frac{1}{2}}^{DS} + \epsilon} \right)^2 \right], \tag{5.45}$$

$m = 0, 1, 2$, *with* $\beta_{m,i\pm\frac{1}{2}}^{DS}$ *defined by* (5.34) *with* (5.35) *and* τ_5 *defined by* (5.30). *Let the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *in* (5.34) *satisfy the Assumption 5.2.1. Then, the resulting WENO-DS method* (5.2) *for smooth solutions of the HCL* (5.1) *exhibits a fifth-order accuracy.*

Proof. From (5.21), we see that the smoothness indicators $\beta_{m,i\pm\frac{1}{2}}$ are of the form

$$\beta_{m,i\pm\frac{1}{2}} = f_x^2 \Delta x^2 + O(\Delta x^4), \tag{5.46}$$

and the global smoothness indicator (5.30)

$$\tau_5 = O(\Delta x^5) \tag{5.47}$$

from (5.31). Then it holds

$$\begin{aligned} \beta_{m,i\pm\frac{1}{2}}^{DS} &= \beta_{m,i\pm\frac{1}{2}} (\delta_{m,i\pm\frac{1}{2}} + C) = \big(f_x^2 \Delta x^2 + O(\Delta x^4) \big) \big(\Phi(\bar{x}_i) + O(\Delta x) + C \big) \\ &= f_x^2 P(\bar{x}_i) \Delta x^2 + O(\Delta x^3). \end{aligned} \tag{5.48}$$

We denote $P(\bar{x}_i) = \Phi(\bar{x}_i) + C$, take $\epsilon = 0$ and set C such that it satisfies the Assumption 5.2.1. Then $P(\bar{x}_i) = O(1)$ is ensured. Then we see that in the non-critical points where $f_x \neq 0$

$$\frac{\tau_5}{\beta_{m,i\pm\frac{1}{2}}^{DS}} = \hat{D} \Delta x^3 + O(\Delta x^4), \tag{5.49}$$

where $\hat{D} = \frac{\left| -\frac{13}{3} f_{xx} f_{xxx} + f_x f_{xxxx} \right|}{f_x^2 P(\bar{x}_i)}$. Substituting this into (5.45) we obtain

$$\alpha_{m,i\pm\frac{1}{2}}^{DS} = d_m \big(1 + O(\Delta x^6) \big) \quad \text{and} \quad \sum_{m=0}^{2} \alpha_{m,i\pm\frac{1}{2}}^{DS} = \big(1 + O(\Delta x^6) \big), \tag{5.50}$$

where we used $\sum_{m=0}^{2} d_m = 1$. It follows directly

$$\omega_{m,i\pm\frac{1}{2}}^{DS} = d_m + O(\Delta x^6) \tag{5.51}$$

and the condition (5.23) is satisfied. Since we ensure $P(\bar{x}_i) > C > \kappa > 0$, the multipliers $P(\bar{x}_i)$ do not introduce any further critical points. Therefore the analysis of the critical points with $f_x = 0$ remains the same as in [13]. Thus the fifth-order accuracy of the WENO-DS scheme with the nonlinear weights (5.45) can be guaranteed also in the critical points. □

Theorem 2. *Let us compute the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *from* (5.34) *in the node* x_i *satisfying* (5.35) *using a one-dimensional convolutional neural network with vector* $\bar{f}(\bar{u}_i)$ *defined by*

$$\begin{aligned} \bar{f}(\bar{u}_i) &= (f(u(x_{i-k})), f(u(x_{i-k+1})), \ldots, f(u(x_{i+k}))), \\ \bar{u}_i = \bar{u}(\bar{x}_i) &= (u(x_{i-k}), u(x_{i-k+1}), \ldots, u(x_{i+k})) \end{aligned} \tag{5.52}$$

as input. Further, let all hidden layers of this neural network be differentiable functions and let the activation function in the last layer of the CNN be bounded from below. Then for these multipliers $\delta_{m,i\pm\frac{1}{2}}$ *the Assumption 5.2.1 holds.*

Proof. Let $2k+1$ be the size of the receptive field of the CNN. Let $F(\cdot) \in \mathbb{R}^{2k+1} \rightarrow \mathbb{R}$ be the convolutional neural network function:

$$F\big(\bar{f}(\bar{u}_i)\big) = \mathrm{CNN}\big(\bar{f}(\bar{u}_i)\big). \tag{5.53}$$

Then, we can define $\Phi = F \circ \bar{f} \circ \bar{u}$ and it holds

$$\begin{aligned} \delta_{0,i+\frac{1}{2}} &= (F \circ \bar{f} \circ \bar{u})(\bar{x}_i - \Delta x) = \Phi(\bar{x}_i - \Delta x), \\ \delta_{1,i+\frac{1}{2}} &= (F \circ \bar{f} \circ \bar{u})(\bar{x}_i) = \Phi(\bar{x}_i), \\ \delta_{2,i+\frac{1}{2}} &= (F \circ \bar{f} \circ \bar{u})(\bar{x}_i + \Delta x) = \Phi(\bar{x}_i + \Delta x). \end{aligned} \tag{5.54}$$

Moreover, as $F, \bar{f}, \bar{u}$ are all differentiable functions, also Φ is differentiable function and (5.38) holds.

As the last activation function is bounded from below, there exists ξ such that $\Phi(\bar{x}_i) > \xi$, and any $C > \kappa + \max(-\xi, 0)$ satisfies the second part of Assumption 5.2.1 for arbitrary $\kappa > 0$. □

5.3 Structure of neural network

In our application, the CNN is used. This is important to ensure spatial invariance of the resulting numerical scheme in a sense that the multipliers δ_m are independent of their position in the spatial grid and just dependent on the solution u itself. This property is not fulfilled for e.g. dense neural network (DNN): the equation (5.53) does not hold with "CNN" replaced by "DNN", as the output of a DNN is dependent on the grid-position index.

The Figure 5.1 shows the values from which the multipliers δ_m, $m = 0, 1, 2$ are constructed, assuming $2k + 1 = 3$ for the receptive field. In this case, the values used to compute the original smoothness indicators are also used to compute the multipliers δ_m, $m = 0, 1, 2$, (see equations (5.20) and (5.29)). If we enlarge the receptive field of the CNN, we also enlarge the stencil for computing the multipliers δ_m, $m = 0, 1, 2$. In this way, the smoothness indicators are basically computed from a wider stencil, which can lead to better numerical approximations. In this case, we just need to add more boundary points before feeding the values (5.52) to the CNN.

We present the whole algorithm of the method in Figure 5.2. This graph corresponds to the computation of the numerical fluxes $\hat{f}^+_{i\pm\frac{1}{2}}$. For the numerical fluxes $\hat{f}^-_{i\pm\frac{1}{2}}$, the index shift in to obtain $\delta_{0,i+\frac{1}{2}}$ and $\delta_{2,i+\frac{1}{2}}$ have to be done reversed according to Figure 5.1.

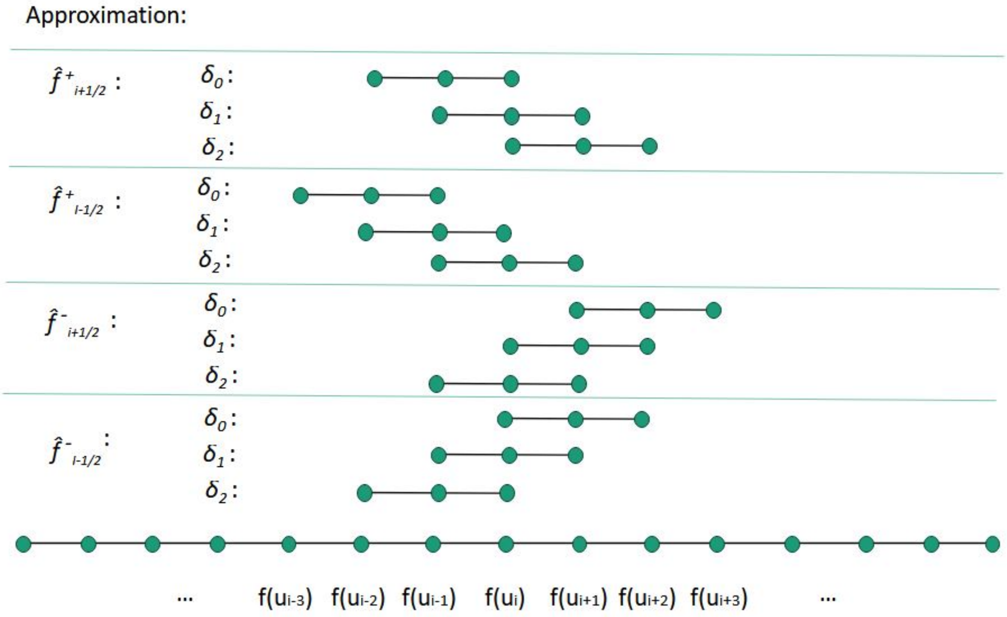


Figure 5.1: The substencils used for computation of multipliers δ_m, $m = 0, 1, 2$ corresponding to the flux approximations $\hat{f}^\pm_{i\pm\frac{1}{2}}$, assuming that for the receptive field of the CNN holds $2k + 1 = 3$.

We use the differentiable activation function ELU for all hidden layers. In the output layer, we use a Softplus activation function, which is bounded from below. The number of the hidden layers, kernel size, and number of channels are chosen separately for each of the equation classes. We move the kernel by one space step so the stride is set to 1. Our goal is to keep the CNN as small as possible, while still achieving the best possible results. In all our experiments, we set $C = 0.1$ in (5.34) and the value of ϵ in (5.45) to 10^{-13}.

Let us note, that this value of C is sufficient. The Softplus activation function generates the values > 0. Therefore the value $C = 0.1$ fulfills the Assumption

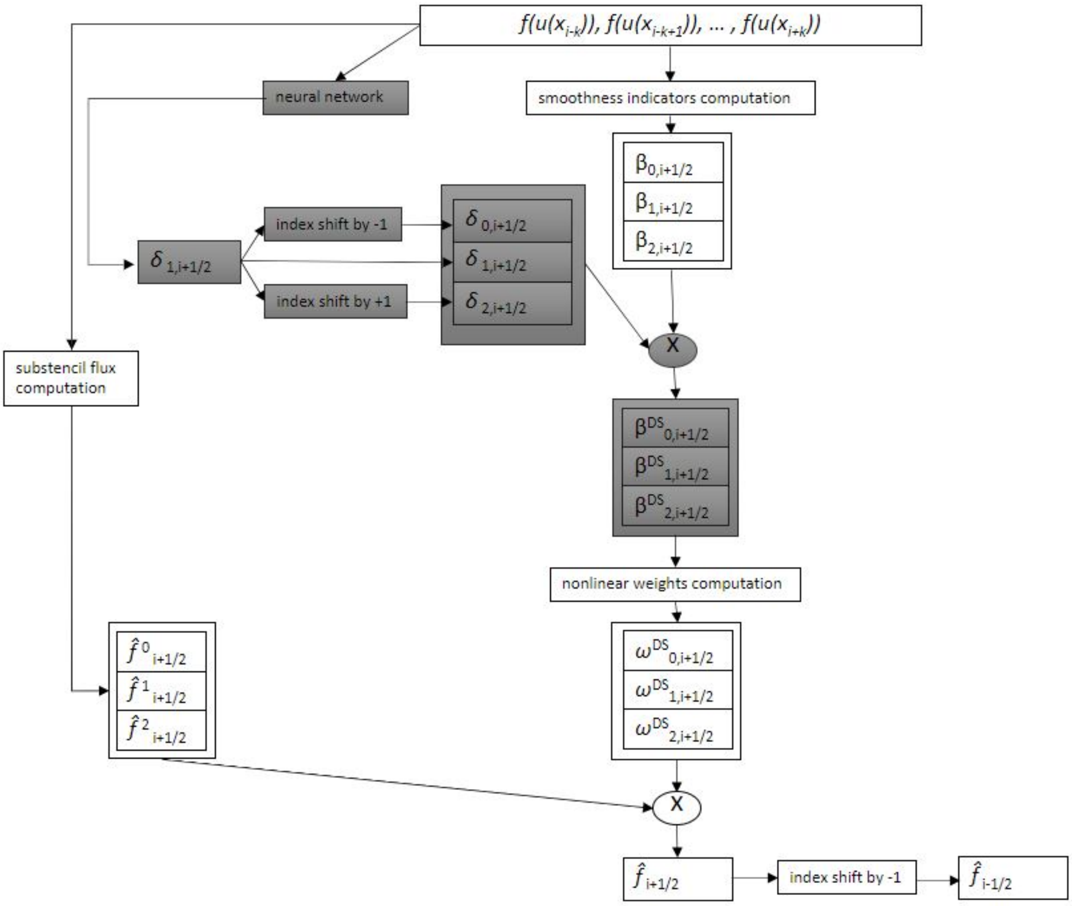


Figure 5.2: The structure of the WENO method combined with the NN algorithm. Original WENO method is represented by the white parts of the graph. The grey parts are added to this method so that the whole graph corresponds to the new method WENO-DS. $2k+1$ is the size of the receptive field of the whole CNN, $\times$ denotes the element-wise multiplication.

5.2.1. For this activation function, we could take any $C > 0$. However, large values of C would decrease the effect of the trained multipliers and make the scheme closer to standard WENO scheme. For small values of C, the experimental order of convergence could be smaller on coarse grids (but still achieved for $\Delta x \to 0$). If another activation function in the output layer would be used, the constant C would have to be adapted so that the Assumption 5.2.1 would still hold.

The proposed NN algorithm can be generally applied to any type of conservation laws. For the equations where discontinuities or shocks are present, we propose to train a CNN separately for each equation class. Then we can better adjust the size of a CNN and its structure as well as the loss function, which leads to better results.

5.4 The training procedure

This section describes how to perform the training procedure. First, we create the dataset for which we compute the reference solutions for the equation (5.1). For the training, we proceed as follows. At the beginning, we randomly choose a problem and its reference solution from our dataset. The weights of the CNN are randomly initialized and we train our model on a solution where our computational domain is divided into $I \times N$ steps, where I is the number of space steps and N is the number of time steps. Then we successively compute the entire solution until the final time T. Using the solution at the time step n, we compute the solution at the time step $n+1$ and during this computation the CNN is used to predict the multipliers of the smoothness indicators. After each of these time steps, we compute the loss and its gradient with respect to the weights of the CNN using backpropagation algorithm.

Let us briefly explain this step. As the WENO method including our deep smoothness modification is differentiable (with respect to the CNN coefficients), we can view the whole WENO scheme, processing inputs in form of solution in time t_n and producing outputs in form of solutions in time t_{n+1}, as a large NN and optimize the parameters of the embedded CNN (used for computing multiplicative factors δ_m) using backpropagation. Following Figure 5.2 representing the WENO-DS scheme, gradients are computed from the bottom to top, until the weights of the CNN are reached. These gradients are used to update the CNN weights. As parameters δ_m depend on these CNN weights, their values also changes in forward path.

To train the network, as in Section 3.3, we use a gradient-based optimizer, namely a variant of stochastic gradient descent, the Adam optimizer [55]. After the last time step at time T, we test a model on a validation set and repeat the above steps. Then we select the model with the best performance on the validation set as our final model.

As the first choice of the loss function we use the mean square error

$$\mathrm{LOSS}_{\mathrm{MSE}}(u) = \frac{1}{I} \sum_{i=0}^{I} (\hat{u}_i^n - u_i^{n,\mathrm{ref}})^2, \tag{5.55}$$

where $\hat{u}_i^n$ is a numerical approximation of $u(x_i, t_n)$ obtained by WENO-DS and $u_i^{n,\mathrm{ref}}$ is the corresponding reference solution. An advantage of this L_2-norm based loss function in contrast to the L_1-norm based loss function is stronger gradients with respect to u_i resulting in faster training.

Let us note that we use for the implementation Python with the deep learning library PyTorch [79].

5.5 Two-dimensional implementation

For two-dimensional implementation of WENO scheme we consider the following two-dimensional form of (5.1):

$$\frac{\partial u}{\partial t} + \frac{\partial f_1(u)}{\partial x} + \frac{\partial f_2(u)}{\partial y} = 0, \qquad t > 0. \tag{5.56}$$

The procedure described in Section 5.1, resp. 5.2 can be easily applied dimension-by-dimension to obtain the approximations of numerical fluxes $\hat{f}_{i+\frac{1}{2},j}$ and $\hat{k}_{i,j+\frac{1}{2}}$, such that it holds

$$\begin{aligned} \frac{1}{\Delta x}\big(\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}\big) &= \big(f_1(u)\big)_x|_{(x_i,y_j)} + O\big(\Delta x^5\big), \\ \frac{1}{\Delta y}\big(\hat{k}_{ij+\frac{1}{2}} - \hat{k}_{i,j-\frac{1}{2}}\big) &= \big(f_2(u)\big)_y|_{(x_i,y_j)} + O\big(\Delta y^5\big), \end{aligned} \tag{5.57}$$

using the uniform grid with nodes (x_i, y_j), $\Delta x = x_{i+1} - x_i$, $\Delta y = y_{j+1} - y_j$, $i = 0, \ldots, I$, $j = 0, \ldots, J$. The corresponding semi-discrete form of (5.56) takes the form

$$\frac{du_{i,j}(t)}{dt} + \frac{\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}}{\Delta x} + \frac{\hat{k}_{i,j+\frac{1}{2}} - \hat{k}_{i,j-\frac{1}{2}}}{\Delta y} = 0, \tag{5.58}$$

where $u_{i,j}(t)$ is the numerical approximation to the point value $u(x_i, y_j, t)$.

In the deep learning approach we could use two-dimensional CNN for training in this case to see if the information from the second dimension can improve the smoothness indicators in the first dimension. However, we only use the training on the one-dimensional data and apply the same trained CNN for two dimensional example. This approach is straightforward, as we use dimension-by-dimension principle and the results using this approach can be found in Section 5.6.3.

5.6 Numerical results

For the system of ODEs resulting from (5.2) we use a third-order TVD Runge-Kutta method [100] given by

$$\begin{aligned} u^{(1)} &= u^n + \Delta t\, L(u^n), \\ u^{(2)} &= \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t\, L(u^{(1)}), \\ u^{n+1} &= \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t\, L(u^{(2)}), \end{aligned} \tag{5.59}$$

where $L = -\frac{1}{\Delta x}(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}})$ and u^n is the solution at the time step n. Note, that the TVD methods have been discussed in Section 4.2.1.

For (5.8) we consider in our examples the *Lax-Friedrichs flux splitting*

$$f^{\pm}(u) = \frac{1}{2}\big(f(u) \pm \alpha u\big), \tag{5.60}$$

where $\alpha = \max\limits_{u} |f'(u)|$.

In all provided tables we show L_∞ and L_2 errors and as 'ratio' we denote the minimum error of the methods WENO-JS and WENO-Z divided by the error of WENO-DS (rounded to 2 decimal points).

5.6.1 The Buckley-Leverett equation

In the first example, we apply our NN algorithm to the Buckley-Leverett equation, which was also considered, for example, in [50, 100, 101]. It is a typical example with a non-convex flux function modeling a two-phase fluid flow in a porous medium [70]. The flux in (5.1) is given by

$$f(u) = \frac{u^2}{u^2 + a(1-u)^2}, \quad -1 \leq x \leq 1, \quad 0 \leq t \leq 0.4, \tag{5.61}$$

where $a < 1$ is a constant that represents the ratio of the viscosities of the two fluids. The initial condition is set as

$$u(x,0) = \begin{cases} 1, & \text{if } \quad -0.5 \leq x \leq 0, \\ 0, & \text{elsewhere} \end{cases} \tag{5.62}$$

and we use periodic boundary conditions.

First, we create the dataset containing 300 reference solutions. To obtain them we compute the solutions for the equation (5.1) with the flux (5.61) and the initial condition (5.62) using WENO-Z method. We randomly generate the parameter a from a uniformly distributed range $[0.05, 0.95]$. We divide the computational domain $[-1, 1]$ into 1024 spatial steps and the solution is computed up to time $T = 0.4$, where the time domain is divided into 8960 time steps.

For both training and comparing the performance of the models, we use the loss function defined as

$$\text{LOSS}(u) = \text{LOSS}_{\text{MSE}}(u) + \text{LOSS}_{\text{OF}}(u), \tag{5.63}$$

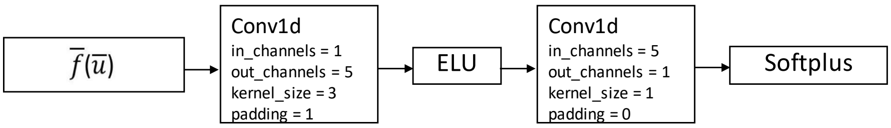
where $\text{LOSS}_{\text{MSE}}(u)$ is defined in (5.55) and

$$\text{LOSS}_{\text{OF}}(u) = \sum_{i=0}^{I} |\min(\hat{u}_i^n, u_{\min}) - u_{\min}| + |\max(\hat{u}_i^n, u_{\max}) - u_{\max}| \tag{5.64}$$

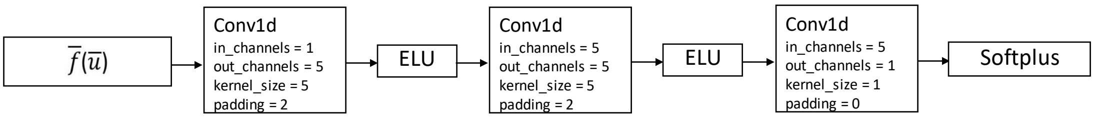
represents the sum of the overflows of the solution above the maximum and below the minimum value of u, in our case $u_{\max} = 1$ and $u_{\min} = 0$. By adding this term to our loss function, we want to avoid the undesirable oscillations that occur especially in the first time steps of the solution.

Size of a neural network and impact on a training procedure.

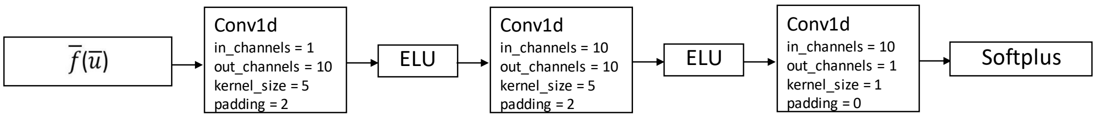
In our implementation, we experimentally find the best CNN for our implementation. Four different CNN structures can be found in Figure 5.3. We try four different structures of a CNN and investigate the impact on a training procedure. First, we use a CNN only with two hidden layers, the CNN shown in Figure 5.3a with a receptive field of size 3. Second, we use 3 hidden layers and increase the size of receptive field with the CNNs shown in Figures 5.3b and 5.3c. Last, we use the CNN with four hidden layers and higher number of channels shown in Figure 5.3d.



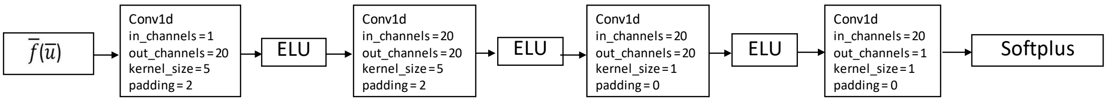
(a) Two hidden layers, lower number of channels, receptive field of size 3.



(b) Three hidden layers, lower number of channels, receptive field of size 9.



(c) Three hidden layers, higher number of channels, receptive field of size 9.



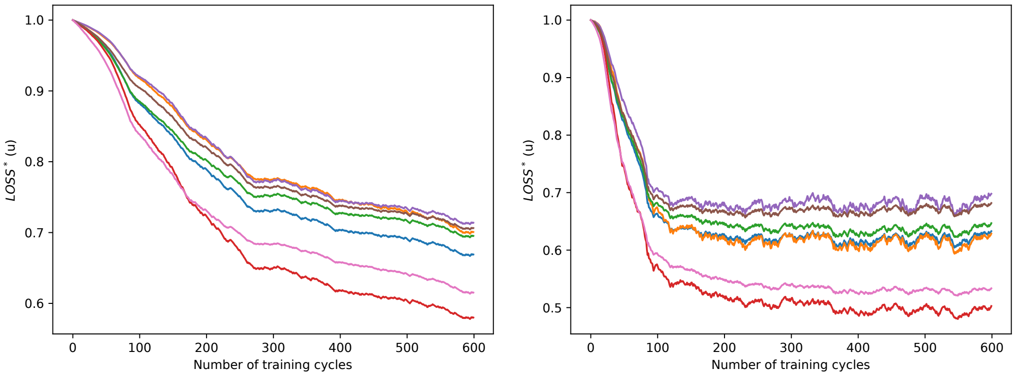
(d) Four hidden layers, higher number of channels, receptive field of size 9.

Figure 5.3: Different structures of the CNN.

For training we fix $I = 128$ and use the Adam optimizer with learning rate 0.0001. In Figures 5.4a-5.4d we show how the value of the loss function for the problems from the validation set (which are not present in the training set) changes with increasing number of training cycles. As training cycle we denote a sequence of training steps performed on a solution for a single randomly chosen parameter a until the final time T. The loss is then evaluated at this final time T. When we plot the loss on validation problems, considering (5.63) we rescale the values for each validation problem to be in the interval $[0,1]$ using the relationship

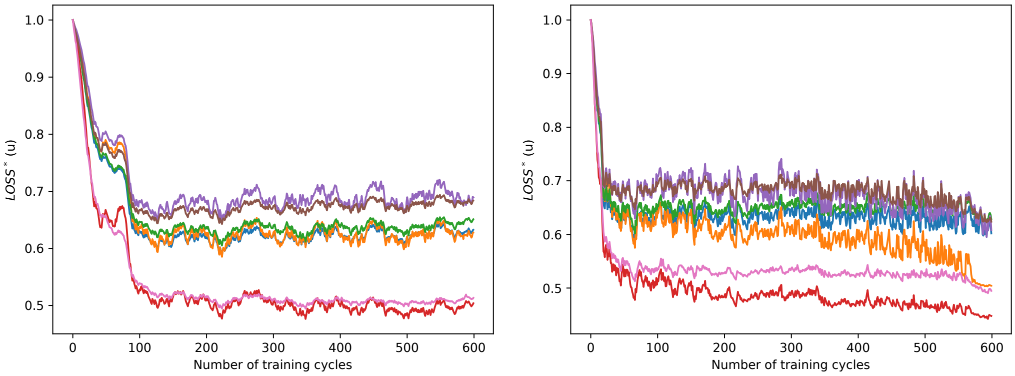
$$\mathrm{LOSS}^*(u) = \frac{\mathrm{LOSS}^l(u)}{\max\limits_{l=0,\dots,L}(\mathrm{LOSS}^l(u))}, \qquad l = 0, \dots, L, \tag{5.65}$$

where L denotes the total number of training cycles.



(a) CNN structure described in Figure 5.3a.

(b) CNN structure described in Figure 5.3b.



(c) CNN structure described in Figure 5.3c.

(d) CNN structure described in Figure 5.3d.

Figure 5.4: Training evolution corresponding to Buckley-Leverett equation: The values (5.65) for different validation problems evaluated after each training cycle.

We run the training for the total number of 600 training cycles. As it can be seen, using a smaller CNN structure we reach the minimal values (5.65) much slower. When we decide, which model we should take as the final model, we compute after each training cycle the sum of the loss values across all validation problems and take the model with its minimal value as our final WENO-DS scheme. For example, using the CNN described in Figure 5.3a we would take the model from one of the latest training cycles. Using the biggest CNN structure from the Figure 5.3c, the optimal model would be after the 223rd training cycle.

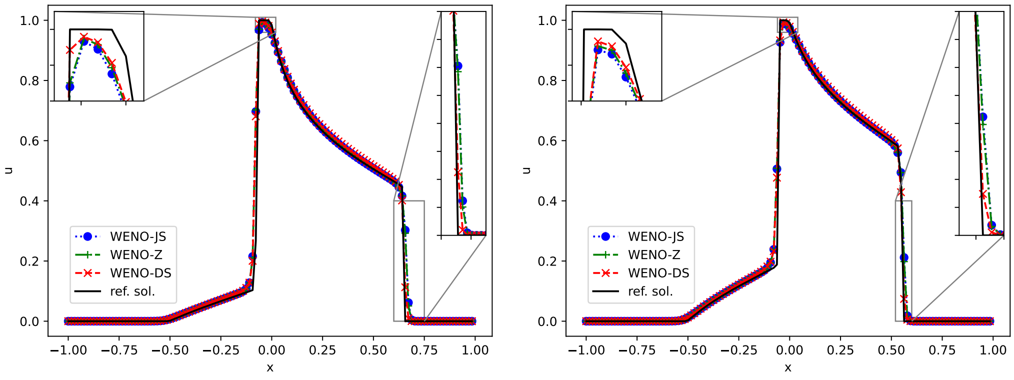
In the following tables, we show the results using model with the CNN structure described in Figure 5.3c (that is from the training which loss evolution is illustrated in Figure 5.4c). The analysis of the actual computational costs will be done in the next part of this section.

We compare the L_∞ and L_2 errors in Table 5.1 for the solution of the conservation law (5.1) with (5.61), $a \in \{0.25, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. These problems create our test set and the listed parameters were neither in the training, nor in the validation set. We highlight the best performing WENO method in bold. We see that WENO-DS outperforms the standard WENO methods in all cases.

	L_∞				L_2			
a	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
0.25	0.429654	0.435090	**0.412758**	1.04	0.068405	0.067912	**0.056616**	1.20
0.4	0.408252	0.405047	**0.301410**	1.34	0.059344	0.058160	**0.045776**	1.27
0.5	0.317824	0.320094	**0.288670**	1.10	0.049913	0.049026	**0.040007**	1.23
0.6	0.459994	0.456687	**0.349394**	1.31	0.062155	0.061275	**0.047238**	1.30
0.7	0.476089	0.475015	**0.372513**	1.28	0.073021	0.072581	**0.058633**	1.24
0.8	0.207676	0.197021	**0.106197**	1.86	0.032560	0.030994	**0.022597**	1.37
0.9	0.375720	0.367802	**0.278328**	1.32	0.062257	0.061834	**0.046312**	1.34

Table 5.1: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Buckley-Leverett equation with the initial condition (5.62), $I = 128$. CNN structure described in Figure 5.3c.

In Figure 5.5 we show the solution of the Buckley-Leverett equation for the test problems with $a = 0.25$ and $a = 0.5$. It can be seen that the WENO-DS gives a better solution quality than the WENO-JS or WENO-Z.



(a) Solution for $a = 0.25$

(b) Solution for $a = 0.5$

Figure 5.5: Comparison of the WENO-JS, WENO-Z and WENO-DS methods on the solution of the Buckley-Leverett equation with the initial condition (5.62), $I = 128$.

Computational cost

We also analyze the computational cost of our method. The additional computational cost of WENO-DS is caused only by the evaluation of the CNN in each computation of the numerical flux approximation. As we still used a rather small

CNN structure, the additional computational costs are not so big, as if we would use big and very complex NN structures.

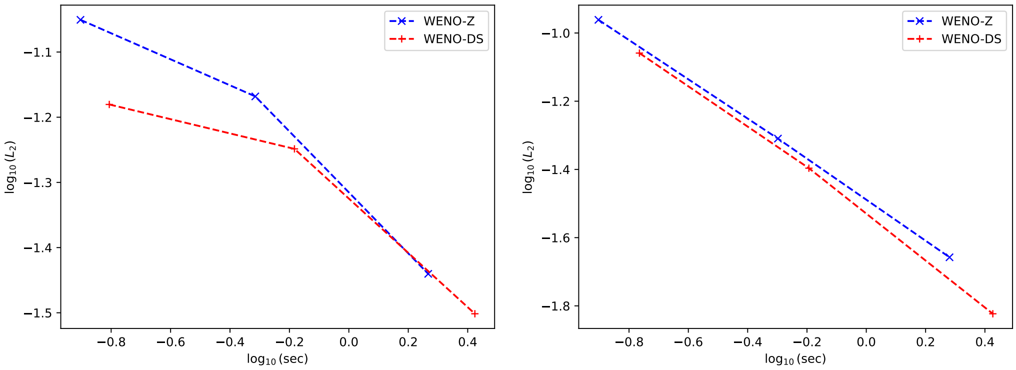
However, a further improvement in the speed of our method could be achieved if the multipliers of the smoothness indicators were updated only once per time step. More precisely, we can compute the multipliers only in the first Runge-Kutta stage and then, assuming that the smoothness of the solution does not change significantly within one time step, the same multipliers could be used in the following two Runge-Kutta stages. This means that the evaluation of the CNN is performed only once per time step instead of three times per time step, and the corresponding additional time cost is reduced to only about 1/3. We used this approach to test our method and examine how the error values change. Corresponding results can be found in Table 5.2. We see that even lower error values are obtained using WENO-DS in most cases, which justifies this approach. Using this approach, we compare the computational costs of WENO-Z and WENO-DS in Figure 5.6 . Let us highlight, that we did not retrain the CNN for different spatial discretizations, that is for $I = 64$ and $I = 256$, which we used in the comparison of computational costs. Furthermore, it should be noted that the speed could be further increased by GPU acceleration.

	L_∞				L_2			
a	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
0.25	0.429654	0.435090	**0.413937**	1.04	0.068405	0.067912	**0.056420**	1.20
0.4	0.408252	0.405047	**0.280662**	1.44	0.059344	0.058160	**0.043520**	1.34
0.5	0.317824	0.320094	**0.288686**	1.10	0.049913	0.049026	**0.040145**	1.22
0.6	0.459994	0.456687	**0.333477**	1.37	0.062155	0.061275	**0.045293**	1.35
0.7	0.476089	0.475015	**0.358451**	1.33	0.073021	0.072581	**0.057281**	1.27
0.8	0.207676	0.197021	**0.106436**	1.85	0.032560	0.030994	**0.023057**	1.34
0.9	0.375720	0.367802	**0.281716**	1.31	0.062257	0.061834	**0.045140**	1.37

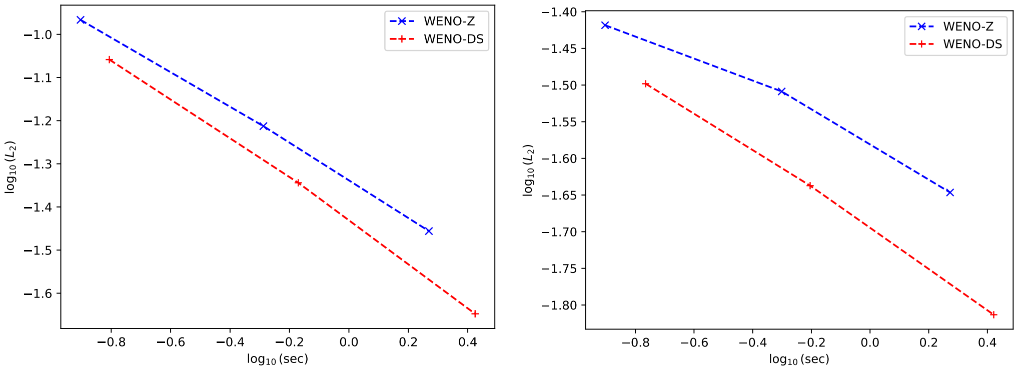
Table 5.2: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Buckley-Leverett equation with the initial condition (5.62), when the evaluation of the CNN was done only in the first Runge-Kutta stage, $I = 128$. CNN structure described in Figure 5.3c.

Let us comment on the additional computational cost caused by the evaluation of the CNN and impact of the size of the CNN on it. Using moderate size of CNN described in Figure 5.3c we obtain WENO-DS method, which remains time-effective in all listed cases from the test set. We compare the compuational costs of WENO-Z and WENO-DS in Figure 5.6. (For clarity we do not plot the compuational costs for WENO-JS, as it is very similar to to WENO-Z and WENO-Z performs in all cases better.) The moderate improvement ratio is obtained for $a = 0.25$ and $a = 0.5$, but as can be seen in Figures 5.6a and 5.6b. the method still remains time efficient.

Using a larger CNN, as described in Figure 5.3d, and with the loss evolution shown in Figure 5.4d, we would obtain a method that could lead to even better improvement ratios, as shown in Table 5.3. As we can see, for example for $a = 0.4$ and $a = 0.6$ we achieve great improvements. On the other hand, for $a = 0.25$ and $a = 0.5$



(a) $a = 0.25$. (b) $a = 0.5$.



(c) $a = 0.6$. (d) $a = 0.8$.

Figure 5.6: Comparison of computational cost against L_2 error on the solution of the Buckley–Leverett equation. CNN structure described in Figure 5.3c, results corresponding to Table 5.2.

the improvement is moderate, and in this case the additional time cost caused by evaluating the larger CNN could not compete with the error improvement. We compare the computational costs in Figure 5.7 and see, that for $a = 0.25$ and for the fine spatial discretization, WENO-Z outperforms WENO-DS. On the other hand, for $a = 0.6$ and $a = 0.8$ WENO-DS performs much better, even when compared to Figure 5.6.

Convergence of WENO-DS on smooth solution.

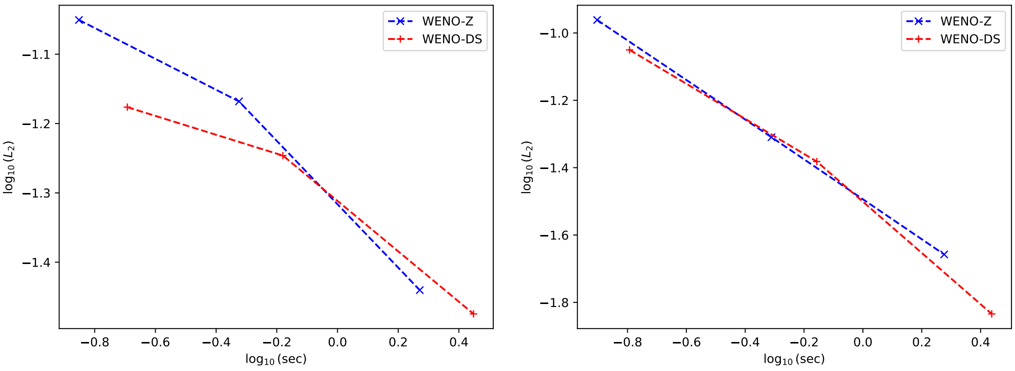
Finally, we verify the analytically proven fifth-order accuracy of the WENO-DS scheme for a transport equation with a smooth solution given as

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \quad u(x,0) = \sin(\pi x), \quad 0 \le x \le 2, \quad 0 \le t \le 0.5, \tag{5.66}$$

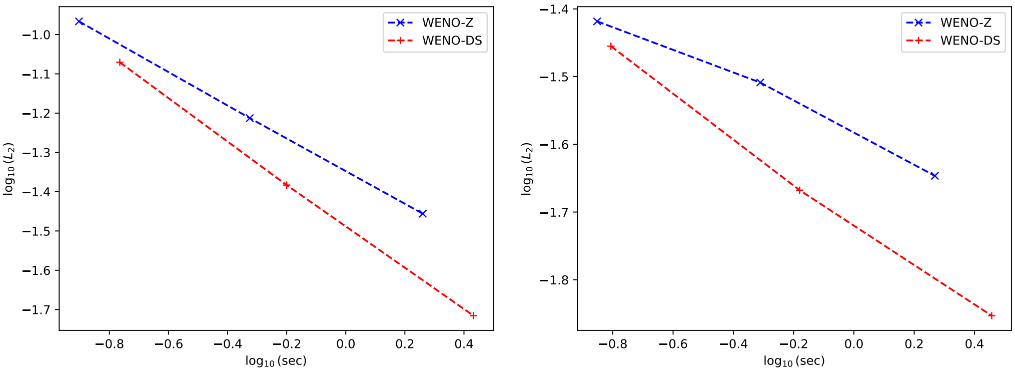
with periodic Dirichlet boundary conditions. Let us note, that we use the same WENO-DS method, which is an output of the training procedure for the Buckley-Leverett equation and we also do not retrain the CNN for different I. Here we demonstrate numerically, that our method can be reliably used also for different class

	L_∞				L_2			
a	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
0.25	0.429654	0.435090	**0.418713**	1.03	0.068405	0.067912	**0.056732**	1.20
0.4	0.408252	0.405047	**0.235125**	1.72	0.059344	0.058160	**0.038288**	1.52
0.5	0.317824	0.320094	**0.300211**	1.06	0.049913	0.049026	**0.041513**	1.18
0.6	0.459994	0.456687	**0.302412**	1.51	0.062155	0.061275	**0.041383**	1.48
0.7	0.476089	0.475015	**0.339466**	1.40	0.073021	0.072581	**0.056251**	1.29
0.8	0.207676	0.197021	**0.103856**	1.90	0.032560	0.030994	**0.021493**	1.44
0.9	0.375720	0.367802	**0.287810**	1.28	0.062257	0.061834	**0.044831**	1.38

Table 5.3: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Buckley-Leverett equation with the initial condition (5.62), when the evaluation of the CNN was done only in the first Runge-Kutta stage, $I = 128$. CNN structure described in Figure 5.3d.



(a) $a = 0.25$. (b) $a = 0.5$.



(c) $a = 0.6$. (d) $a = 0.8$.

Figure 5.7: Comparison of computational cost against L_2 error on the solution of the Buckley–Leverett equation. CNN structure described in Figure 5.3d, results corresponding to Table 5.3.

of equation with different initial condition and remains convergent. The results can be found in Table 5.4. There is a great improvement when we compare our scheme with the WENO-NN scheme of Stevens and Colonius [105], where the resulting scheme exhibits only first-order accuracy.

	WENO-Z		WENO-DS		WENO-Z		WENO-DS	
I	L_∞	Order	L_∞	Order	L_2	Order	L_2	Order
20	9.369742e-03	-	1.021138e-02	-	9.401362e-03	-	9.470170e-03	-
40	2.558719e-04	5.194516	2.567739e-04	5.313535	2.560057e-04	5.198622	2.560359e-04	5.208972
80	9.466151e-06	4.756500	9.467514e-06	4.761369	9.472722e-06	4.756253	9.472750e-06	4.756419
160	3.177833e-07	4.896663	3.177833e-07	4.896871	3.178113e-07	4.897537	3.178113e-07	4.897541
320	9.957350e-09	4.996137	9.957350e-09	4.996137	9.957437e-09	4.996252	9.957437e-09	4.996252
640	3.117835e-10	4.997145	3.117830e-10	4.997147	3.117850e-10	4.997151	3.117851e-10	4.997150

Table 5.4: L_∞ and L_2 errors with convergence order of WENO-Z and WENO-DS on equation (5.66).

Although our scheme remains convergent for any type of equation and for arbitrary discretization, in examples with strong discontinuities we recommend to retrain the NN for solving a new class of PDE. Doing so, we can improve the performance of the NN and achieve the enhancement when compared to the existing methods. We refer to [105], where authors aim to use the NN trained only once for any class of PDE. However, no improvement e.g. in Burgers' equation is achieved and as demonstrated in the example with one-dimensional Euler equations, also no improvement can be guaranteed when the discretization changes.

5.6.2 The inviscid Burgers' equation

In the next example we consider the inviscid Burgers' equation, where the flux function in (5.1) is given by

$$f(u) = \frac{u^2}{2}, \quad 0 \leq x \leq 2, \quad 0 \leq t \leq 0.3. \tag{5.67}$$

We consider following initial conditions

$$u(x,0) = \begin{cases} z_1, & \text{if} \quad 1 \leq x \leq 2, \\ 0, & \text{elsewhere}, \end{cases} \tag{5.68}$$

$$u(x,0) = \exp\big(-z_2(x-1)^2\big), \tag{5.69}$$

$$u(x,0) = z_3 \sin(\pi x), \tag{5.70}$$

where

$$z_1 \in \mathcal{U}[1,2], \quad z_2 \in \mathcal{U}[10,30], \quad z_3 \in \mathcal{U}[1,2]. \tag{5.71}$$

Using these initial conditions, we cover problems with both continuous and discontinuous initial data, and we simulate the shocks and discontinuities very well. We train a single CNN on all mentioned classes of initial conditions and use periodic Dirichlet boundary conditions.

We first create the data set for training consisting of 300 reference solutions, in which we compute these solutions of the Burgers' equation with the initial conditions (5.68)–(5.70). The computational domain is divided into 1024 space steps, the time step is chosen such that $0.4/\Delta t = \max_u |f'(u)|/\Delta x$ and the solution is computed up to time $T = 0.3$ using the WENO-Z scheme.

Also in this example, we experimentally find the best CNN structure with the best performance on the validation set. As we aim to cover more different problems with various initial conditions, we use the bigger CNN structure illustrated in Figure 5.3d. For this case, we experimentally found out, that this CNN has better ability to generalize for various initial value problems. Using smaller CNN, it could happen, that we would achieve the significant improvement for one of the initial conditions (5.68)-(5.70) and for another ones, only very small error improvement. For training we use the Adam optimizer with learning rate 0.0001, fix $I = 128$ and proceed as described in Section 5.4.

During training we observe the different magnitude of the loss values (5.55) for different problems from the training set To match the training contribution from very small loss problems to large loss problems, we use the following adapted loss function:

$$\mathrm{LOSS}_{\mathrm{AD}}(u) = \frac{1}{10} 10^{|\lceil \log_{10}(\mathrm{LOSS}_{\mathrm{MSE}}(u)) \rceil|} \, \mathrm{LOSS}_{\mathrm{MSE}}(u), \tag{5.72}$$

where $\mathrm{LOSS}_{\mathrm{MSE}}(u)$ is defined in (5.55). Let us note, that there is no gradient propagation through the exponential part in the equation (5.72) as the ceiling operation $\lceil v \rceil$, giving as output the least integer greater than or equal to v, is step-wise constant. As on the previous example, we choose the model with the best performance on validation set and present the results in following tables and figures.

Let us note, that during training we update the multipliers of the smoothness indicators in each Runge-Kutta stage. According to results for the Buckley-Leverett equation, when we test the method, we again update the multipliers only once per time step, namely in the first Runge-Kutta stage. For another Runge-Kutta stages we use the same multipliers.

We compare the errors on the problems from the test set in Tables 5.5 and 5.6. These were not in the training or validation set and the parameters were randomly generated. It should be noted that although our training set was created with the parameters sampled from uniform distribution as specified in (5.71), the method can also generalize for parameter values outside of these intervals, as can be seen in Table 5.6. We observe great improvement for problems with initial condition (5.70) for the parameters inside, as well as outside of training set intervals. For problems with the initial condition (5.68) and (5.69) we observe rather moderate error improvement, but as it is illustrated in Figure 5.8, WENO-DS is able to capture shocks and discontinuities very well.

In Figure 5.8 we show the solution of the Burgers' equation with the initial condition (5.68) for $z_1 = 1.84$, (5.69) for $z_2 = 29.08$, (5.70) for $z_3 = 1.6$ and (5.70) for $z_3 = 3.0$. We observe that WENO-DS captures shocks and discontinuities very well and gives us a better quality of solution compared to WENO-JS and WENO-Z.

Next, we test our method on four examples, where the Burgers' equation with the

		L_∞				L_2			
initial condition	z_j	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
(5.68)	1.19	0.665675	0.667148	**0.647964**	1.03	0.086363	0.086253	**0.082829**	1.04
	1.53	0.595493	0.587766	**0.537820**	1.09	0.080244	0.078715	**0.070645**	1.11
	1.84	0.744080	0.736504	**0.675833**	1.09	0.099022	0.097493	**0.087989**	1.11
(5.69)	14.94	0.114671	0.105817	**0.100248**	1.06	0.017054	0.015293	**0.015133**	1.01
	21.65	0.236526	0.229823	**0.202748**	1.13	0.032984	0.031699	**0.027131**	1.17
	29.08	0.310989	0.309104	**0.300668**	1.03	0.040377	0.039908	**0.037727**	1.06
(5.70)	1.46	0.058185	0.055862	**0.013055**	4.28	0.010327	0.009918	**0.002244**	4.42
	1.6	0.062979	0.060594	**0.011377**	5.33	0.013107	0.012751	**0.007136**	1.79
	1.9	0.074034	0.071443	**0.028508**	2.51	0.013272	0.012820	**0.004949**	2.59

Table 5.5: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Burgers' equation with the initial condition parameters inside of training set intervals (5.71).

		L_∞				L_2			
initial condition	z_j	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
(5.68)	0.71	0.220244	0.215243	**0.189543**	1.14	0.033159	0.032157	**0.027803**	1.16
	2.57	1.026243	1.014877	**0.931102**	1.09	0.135856	0.133667	**0.120652**	1.11
	2.8	0.762665	0.743593	**0.644627**	1.15	0.113972	0.110324	**0.095163**	1.16
(5.69)	35.5	**0.111681**	0.111857	0.120455	0.93	0.017291	0.016854	**0.015420**	1.09
	33.9	0.352996	0.347349	**0.324168**	1.07	0.045858	0.044906	**0.041029**	1.09
	34.67	0.289670	0.287470	**0.270236**	1.06	0.037770	0.037282	**0.034038**	1.10
(5.70)	2.12	0.084515	0.081753	**0.038086**	2.15	0.016966	0.016537	**0.009930**	1.67
	2.44	0.084270	0.081280	**0.035242**	2.31	0.014942	0.014415	**0.005396**	2.67
	3.0	0.083943	0.080915	**0.034467**	2.35	0.014840	0.014305	**0.005153**	2.78

Table 5.6: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Burgers' equation with the initial condition parameters outside of training set intervals (5.71).

flux function (5.67) and following initial conditions will be solved:

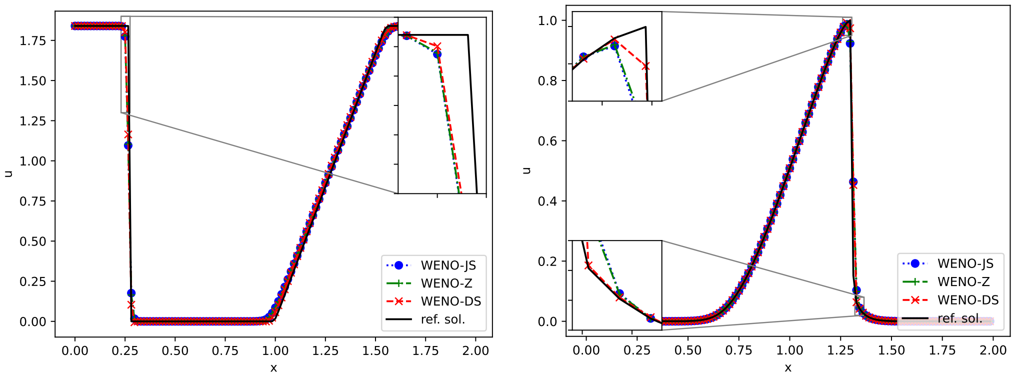
$$u(x,0) = 1 + \sin(4\pi x), \quad 0 \le x \le 2, \tag{5.73}$$

$$u(x,0) = 2\sin(4\pi x), \quad 0 \le x \le 2, \tag{5.74}$$

$$u(x,0) = 1.5\cos(\pi x), \quad 0 \le x \le 2, \tag{5.75}$$

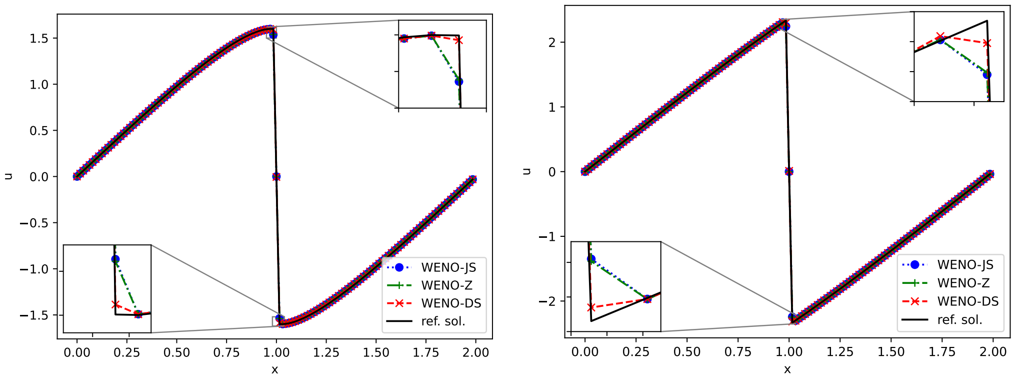
$$u(x,0) = \sin(2\pi x), \quad 0 \le x \le 2. \tag{5.76}$$

We compute the solution up to time $T = 0.3$. Let us note, that the method was not retrained on these initial conditions and still performs well. We compare L_∞ and L_2 errors in Table 5.7 and observe great improvement achieved by WENO-DS method. Figure 5.9 illustrates the solution. As reference solution, we use the solution computed by WENO-Z method on a fine space domain divided into 1024 space steps.



(a) Initial condition (5.68) with $z_1 = 1.84$.

(b) Initial condition (5.69) with $z_2 = 29.08$.



(c) Initial condition (5.70) with $z_3 = 1.6$.

(d) Initial condition (5.70) with $z_3 = 3.0$.

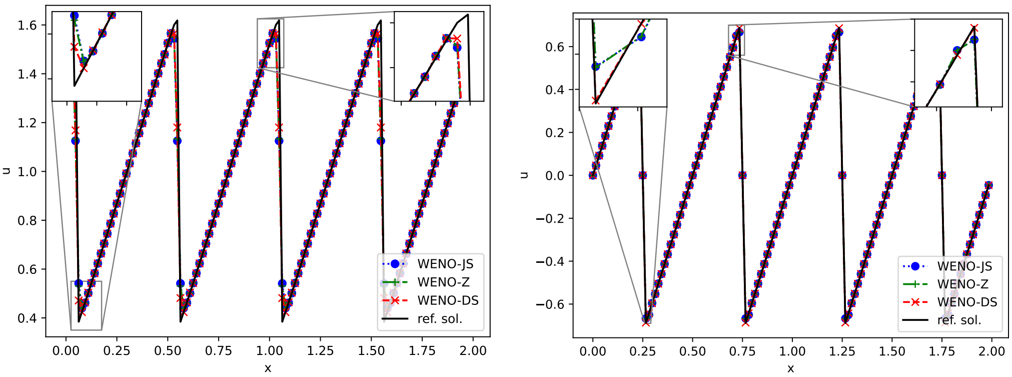
Figure 5.8: Comparison of the WENO-JS, WENO-Z and WENO-DS methods for the solution of the Burgers' equation with various initial conditions, $I = 128$.

	L_∞				L_2			
initial condition	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
(5.73)	0.49317	0.484954	**0.450177**	1.08	0.130203	0.127337	**0.113196**	1.12
(5.74)	0.021023	0.020494	**0.002063**	9.93	0.007973	0.007754	**0.000705**	11.00
(5.75)	0.060176	0.057868	**0.011701**	4.95	0.010638	0.01023	**0.001955**	5.23
(5.76)	0.035934	0.034601	**0.006844**	5.06	0.009009	0.00868	**0.001736**	5.00

Table 5.7: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Burgers' equation with the initial conditions (5.73)-(5.76).

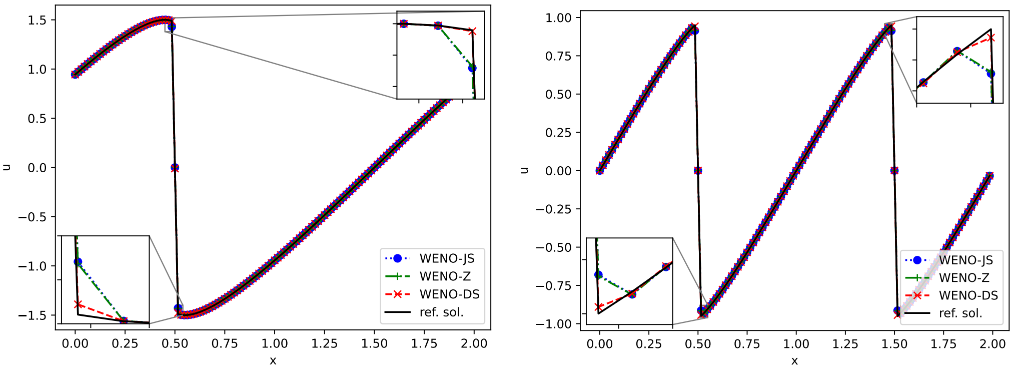
5.6.3 The two-dimensional Burgers' equation

To demonstrate the performance of WENO-DS in two-dimensional space we apply the method trained on one-dimensional data for the Burgers' equation with the flux



(a) Initial condition (5.73).

(b) Initial condition (5.74).



(c) Initial condition (5.75).

(d) Initial condition (5.76).

Figure 5.9: Comparison of the WENO-JS, WENO-Z and WENO-DS methods for the solution of the Burgers' equation with various initial conditions, $I = 128$.

function (5.67) to the two-dimensional Burgers' equation of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} + \frac{\partial f(u)}{\partial y} = 0, \qquad f(u) = \frac{u^2}{2} \tag{5.77}$$

on the spatial domain $[-1, 1] \times [-1, 1]$ divided into 128×128 uniform cells. As considered by Cao, Xu and Zheng [14] we use the initial condition

$$u(x, y, 0) = (x^2 - 1)^2\,(y^2 - 1)^2, \qquad -1 \leq x \leq 1, \quad -1 \leq y \leq 1. \tag{5.78}$$

We present the solution at time $T = 0.8$ in Figure 5.10. Corresponding L_∞ and L_2 errors for 64×64 spatial discretization can be found in Table 5.8. The reference solution was computed on the spatial discretization with 256×256 cells. Let us emphasize, that no additional retraining was needed, as we apply the method using dimension-by-dimension principle.

This example demonstrates, that the WENO-DS method trained on one-dimensional data can be effectively used also for two-dimensional problems, providing better quality of solution.

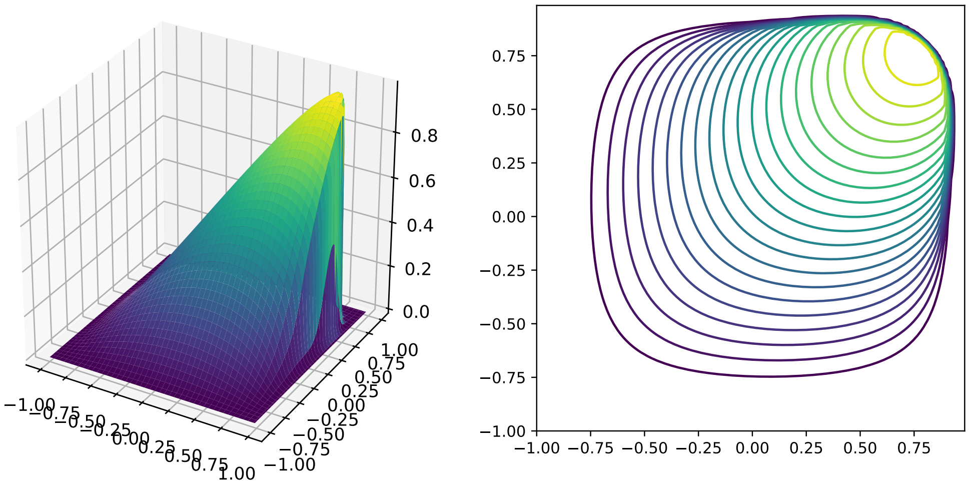


Figure 5.10: Numerical solution of the two-dimensional Burgers' equation using WENO-DS at $T = 0.8$, 128×128 cells.

	L_∞				L_2			
$I \times J$	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
64×64	0.409603	0.408364	**0.350585**	1.16	0.023479	0.022834	**0.019379**	1.18

Table 5.8: Comparison of L_∞ and L_2 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the two-dimensional Burgers equation with the initial condition (5.78).

6

Chapter 6

Deep smoothness WENO scheme for Euler system

It has long been a challenge to adequately simulate complex flow problems such as the Euler equations using numerical methods. In this chapter we extend the WENO-DS approach introduced in the previous chapter to solving a general one-dimensional and two-dimensional Euler system of gas dynamics. Through intensive study of numerous test problems, which involve various shocks and rarefaction waves, the new technique is shown to outperform traditional fifth-order WENO schemes, especially in cases where the numerical solutions exhibit excessive diffusion or overshoot around shocks. Recall that the terms shock wave, rarefaction wave and contact discontinuity were introduced in Section 4.1.

6.1 One-dimensional Euler system

Let us introduce the one-dimensional Euler system of the form

$$U_t + F(U)_x = 0, \tag{6.1}$$

with

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \qquad F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E+p) \end{pmatrix} \tag{6.2}$$

for polytropic gas. Here, the variable ρ is the density, u the velocity component, p the pressure and E the total energy given by

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho u^2. \tag{6.3}$$

We take $\gamma = 1.4$ in our implementation, which is the ratio of the specific heats.

To obtain the WENO approximations in the one-dimensional example, we apply the procedure described in Section 5.1. Thus, we obtain the flux approximation for (6.1) as

$$\frac{1}{\Delta x}\big(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\big) = \big(F(U)\big)_x\big|_{(x_i)} + O\big(\Delta x^5\big), \tag{6.4}$$

with the uniform grid defined by $\Delta x = x_{i+1} - x_i$, $i = 0, \ldots, I$.

Further, the deep learning approach described in Section 5.2 is applied. That means, the new smoothness indicators

$$\begin{aligned}\beta^{DS}_{m,i+\frac{1}{2}} &= \beta_{m,i+\frac{1}{2}}(\delta_{m,i+\frac{1}{2}} + C),\\ \beta^{DS}_{m,i-\frac{1}{2}} &= \beta_{m,i-\frac{1}{2}}(\delta_{m,i-\frac{1}{2}} + C),\end{aligned} \tag{6.5}$$

are inserted to original WENO-Z scheme from Section 5.1.3 which is improved to the WENO-DS scheme. For the construction of the WENO-DS scheme the nonlinear weights (5.45) are used.

In our examples, we proceed with the implementation of the Euler system using characteristic decomposition. This means that we first project the solution and the flux onto the characteristic fields using left eigenvectors. Then we apply the Lax-Friedrichs flux splitting (5.60) for each component of the characteristic variables. These values are fed into the CNN and the enhanced smoothness indicators are computed. After obtaining the final WENO approximation, the projection back to physical space is done using the right eigenvectors, cf. [102] for more details.

Remark 4. The deep learning approach described in Section 5.2 remains the same. However, one adjustment needs to be done as we use the characteristic-wise implementation in this case and the matrix of eigenvectors is computed for a frozen average state at the cell-face. That means, the equation (5.35) needs to be modified, such that for the multipliers holds

$$\begin{aligned}\delta_{0,i-\frac{1}{2}} &= \Phi(\bar{x}_i - 2\Delta x),\\ \delta_{0,i+\frac{1}{2}} &= \Phi(\bar{x}_i - \Delta x) = \delta_{1,i-\frac{1}{2}},\\ \delta_{1,i+\frac{1}{2}} &= \Phi(\bar{x}_i) = \delta_{2,i-\frac{1}{2}},\\ \delta_{2,i+\frac{1}{2}} &= \Phi(\bar{x}_i + \Delta x).\end{aligned} \tag{6.6}$$

Analogously, for the negative part of the flux splitting, the equation (5.37) has to be modified such that

$$\begin{aligned}\delta_{0,i+\frac{1}{2}} &= \Phi(\bar{x}_i + 2\Delta x),\\ \delta_{0,i-\frac{1}{2}} &= \Phi(\bar{x}_i + \Delta x) = \delta_{1,i+\frac{1}{2}},\\ \delta_{1,i-\frac{1}{2}} &= \Phi(\bar{x}_i) = \delta_{2,i+\frac{1}{2}},\\ \delta_{2,i-\frac{1}{2}} &= \Phi(\bar{x}_i - \Delta x).\end{aligned} \tag{6.7}$$

The NN function $\Phi(\cdot)$ satisfies the Assumption 5.2.1 and is defined as in Theorem 2. This means, that the accuracy analysis described in Section 5.2.1 holds. The Figure 5.1 explanatory illustrates the values from which the multipliers δ_m, $m = 0, 1, 2$ are constructed.

6.1.1 Structure of neural network and the training procedure

For the training of the WENO-DS method we use a CNN. We have experimented with different training procedures and introduce in this chapter a novel and most effective one, which gives the best numerical results.

First, the data set must be created. At the beginning of training, the weights of the CNN are randomly initialized and a problem is selected from the data set. Then we perform one time step and use the CNN to predict the multipliers of the smoothness indicators. Then we compute the loss and its gradient with respect to the weights of the CNN using the backpropagation algorithm. For more details explaining the backpropagation procedure we refer to Section 5.4.

After this step, we do not automatically proceed to the next time step (as it was described in Section 5.4), but randomly decide whether to proceed to the next time step of a current problem or select another problem from our data set and run one time step of that problem. The probability of choosing the new problem is determined at the beginning of the training session. We use the probability $\varphi = 0.5$ in our experiments. This means that we select a new problem from a data set with probability $\varphi = 0.5$. We set the maximum number of opened problems to 200. We remember all opened problems, and if no new problem is opened (with probability $1 - \varphi$), we execute the next time step of a problem uniformly chosen from the set of already opened problems. When the maximum number of opened problems is reached, we successively compute the entire solution for these problems until the final time T. After reaching T for some of these problems, this problem is closed and another one can be opened.

After each of these time steps, the loss and its gradient are calculated with respect to the weights of the CNN. The gradient is then used to update the weights. Keeping the solution from the previous time step as initial data, we repeat the same procedure until we reach the maximum number of training steps. This training procedure gives us a great opportunity to mix the solutions with different initial data and in different time points, which makes the training more effective.

To train the network, we use the Adam optimizer [55]. The learning rate is set to 0.0001 to update the weights of the CNN. This near-optimal learning rate was found through experiments.

Adaptive activation functions

We can make the training more effective and get better numerical results by using *adaptive activation functions* [47, 48, 49]. They can adapt to the problem at hand. In this work, we used global adaptive activation functions [47], where the additional slope parameter is introduced in the activation function as follows.

For the ELU activation function, we train the additional parameter α:

$$\mathrm{ELU} = \begin{cases} x, & \text{if} \quad x > 0, \\ \alpha(\exp(x) - 1) & \text{if} \quad x \leq 0, \end{cases} \tag{6.8}$$

and we denote the adaptive ELU as *aELU*. For the Softplus activation function, we train the additional parameter β:

$$\mathrm{Softplus}(x) = \frac{1}{\beta}\log(1 + \exp(\beta x)), \tag{6.9}$$

and we denote the adaptive Softplus as *aSoftplus*.

Loss function

In this work, the loss function consists of the data mismatch term between the solution predicted by the CNN and the reference solution. For the loss function, we use the mean square error loss as follows:

$$LOSS_{\mathrm{MSE}}(u) = \frac{1}{I}\sum_{i=0}^{I}(\hat{u}_i - u_i^{\mathrm{ref}})^2, \tag{6.10}$$

where $\hat{u}_i$ is a numerical approximation of $u(x_i)$ and u_i^{ref} is the corresponding reference solution. However, in our examples, we use the L_1 norm as the main error measure, which is more typical for hyperbolic problems. Thus, for validation during the training, we use the metrics

$$L_1(u) = \frac{1}{I}\sum_{i=0}^{I}|\hat{u}_i - u_i^{\mathrm{ref}}|. \tag{6.11}$$

The reference solution are computed according to [115].

For the Euler system (6.1) - (6.2) we have to adapt the loss function from (6.10) and use it for training

$$LOSS_{\mathrm{MSE}}(\rho, u, p) = LOSS_{\mathrm{MSE}}(\rho) + LOSS_{\mathrm{MSE}}(u) + LOSS_{\mathrm{MSE}}(p), \tag{6.12}$$

and for the validation during training from (6.11) we use

$$L_1(\rho, u, p) = L_1(\rho) + L_1(u) + L_1(p). \tag{6.13}$$

The pseudo-code of the whole training procedure is described in Algorithm 2

Algorithm 2 WENO-DS training procedure

```
for $l \leftarrow 0$ to $L$ do                    ▷ $L$: the total number of training steps
    → With probability $\varphi$ choose a new problem from a data set and store this
    problem and with probability $1-\varphi$ choose a problem from the set of already
    opened problems
    Input: $\big(F(U(x_0)) \quad F(U(x_1)) \quad \dots \quad F(U(x_I))\big)$ → Evaluation of CNN
    Output: $\Big(\delta_{1,0+\frac{1}{2}} \quad \delta_{1,1+\frac{1}{2}} \quad \cdots \quad \delta_{1,I+\frac{1}{2}}\Big)$     ▷ prediction of CNN
    → From these values, obtain the values $\delta_{0,i+\frac{1}{2}}$ and $\delta_{2,i+\frac{1}{2}}$, $i = 0, \dots, I$ using the
    relations (6.6) and (6.7)
    → Compute loss function (6.10)
    → Compute loss gradient with respects to the weights of CNN using backpropagation
            ▷ Following Figure 5.2 representing the WENO-DS scheme, gradients are
    computed from the bottom to top, until the weights of the CNN are reached
    → Update weights of CNN using chosen optimizer
end for
```

Size of the neural network

Determining the best training procedure and the most suitable CNN structure often requires experimentation. It is important to find the optimal balance between the length of the training process, the size of the NN and the actual execution time of the NN (during the computation of the PDE). Although the training process is only done as *offline* training, i.e. it only needs to be done once, we aim to do it in a reasonable amount of time. On the one hand, we could use a very complex NN structure, which could potentially converge to the optimal WENO-DS very quickly. This means that offline training would be very short. However, the actual computational time of the method would be higher and the method would be inefficient. On the other hand, we could use very small NNs. In this case, however, offline training could take much longer. We focus on finding the smallest possible NN that has a reasonable offline training time and still remains computationally efficient compared to WENO-Z.

In Figure 6.1 we describe the CNN structure used for a training for the one-dimensional Euler system. We have 3 input channels in the first hidden layer and 3 output channels in the last hidden layer. These represent the dimension of the solution U from (6.1). In this way, the CNN also incorporates information from other variables, which can be useful for improving the numerical solution. The input $\bar{F}(\bar{U})$ represents the numerical approximation after left eigenvector projection and flux splitting.

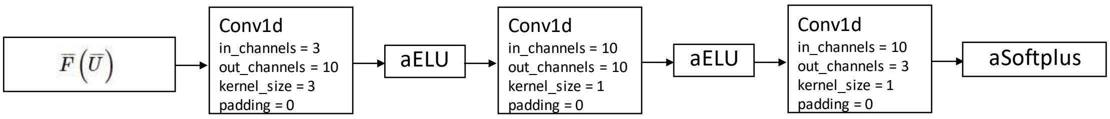


Figure 6.1: A structure of the CNN used for the one-dimensional Euler system.

Construction of the data set for the CNN training procedure

In presented examples the solution of the Euler system consists of the left rarefaction wave, the right travelling contact wave and the right shock wave. We want to imitate this behavior of the solution, so we construct our data set as described in Algorithm 3.

Algorithm 3 Generation of parameters for the data set for one-dimensional Euler equations of gas dynamics

```
Choose randomly $s \in \{0, 2\}$
if $s = 0$ then
    $p_l = a + b, \quad a \in \mathcal{U}[0.5, 1.5], \quad b \in \mathcal{U}[-0.05, 0.05],$
    $p_r = 1/c, \quad c \in \mathcal{U}[5, 10],$
    $\rho_l = p_l,$
    $\rho_r = p_r + d, \quad d \in \mathcal{U}[-0.05, 0.05],$
    $u_l = e, \quad e \in \mathcal{U}[0.5, 1],$
    $u_r = 0,$
else if $s = 1$ then
    $p_l = 1,$
    $p_r = 0.1,$
    $\rho_l = k, \quad k \in \mathcal{U}[1, 2],$
    $\rho_r = \frac{1}{10}\rho_l + l, \quad l \in \mathcal{U}[-0.05, 0.05],$
    $u_l = m, \quad m \in \mathcal{U}[0.5, 1],$
    $u_r = 0,$
else
    $p_l = n, \quad n \in \mathcal{U}[3, 4],$
    $p_r = \frac{1}{7}p_l + q, \quad q \in \mathcal{U}[-0.05, 0.05]$
    $\rho_l = r, \quad r \in \mathcal{U}[0.3, 0.6],$
    $\rho_r = \rho_l + s, \quad s \in \mathcal{U}[-0.05, 0.05],$
    $u_l = t, \quad t \in \mathcal{U}[0.5, 1],$
    $u_r = 0,$
end if
```

where

$$(\rho, u, p) = \begin{cases} (\rho_l, u_l, p_l) & 0 \le x < 0.5, \\ (\rho_r, u_r, p_r) & 0.5 \le x \le 1. \end{cases} \tag{6.14}$$

6.1.2 Numerical examples

To solve the following system of ODEs

$$\frac{dU(t)}{dt} = L(U), \tag{6.15}$$

we use a third-order TVD Runge-Kutta method according to (5.59). In our application, to obtain the best possible results, we evaluate the CNN during training, as well as during testing, in each Runge-Kutta stage.

For the temporal discretization we use an adaptive step size

$$\Delta t = \frac{0.9\Delta x}{\max(c_i + |u_i|)}, \quad c^2 = \frac{\gamma p}{\rho}, \tag{6.16}$$

where u_i is the local velocity and c_i the local speed of sound.

The training procedure, as well as structure of the CNN was already discussed in Section 6.1.1. During training and validation, the spatial domain is divided uniformly into 100 space steps and the final time T is fixed to 0.1. We run the training for the total number of 10000 training steps. Using the relationship

$$L_1^*(\rho, u, p) = \frac{L_1^l(\rho, u, p)}{\max\limits_{l=0,\dots,L}(L_1^l(\rho, u, p))}, \qquad l = 0, \dots, L, \tag{6.17}$$

we rescale validation metrics values and show their evolution for the validation problems in Figure 6.2. We see that for most of the validation problems, the loss is decreasing with increasing number of training steps. However, after some number of training steps, it increases significantly for some of the problems. We select the model from the training step 6300 as our final WENO-DS scheme. This is a training step, in which the sum of the validation values (6.13) over all validation problems obtains its lowest value.

The most common benchmark problems are the Lax problem [67] with an initial condition

$$(\rho, u, p) = \begin{cases} (0.445, 0.698, 3.528) & 0 \le x < 0.5, \\ (0.5, 0, 0.571) & 0.5 \le x \le 1, \end{cases} \tag{6.18}$$

and the Sod problem [104], where the initial condition is specified as

$$(\rho, u, p) = \begin{cases} (1, 0.75, 1) & 0 \le x < 0.5, \\ (0.125, 0, 0.1) & 0.5 \le x \le 1. \end{cases} \tag{6.19}$$

In these and further examples, Dirichlet boundary conditions are used. We present the solution of the Lax problem (6.18) for ρ, u and p with the final time $T = 0.13$ in Figure 6.3 and the corresponding L_1 error values can be found in Table 6.1. In this table the error values for $I = 100, 200$ and 1000 can be found. As 'ratio' we denote

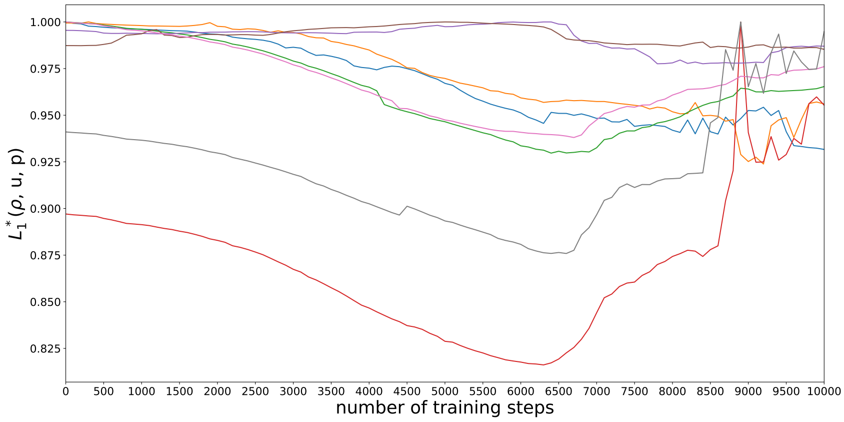


Figure 6.2: Training evolution corresponding to one-dimensional Euler system: The values (6.17) for different validation problems evaluated each 100 training steps.

the minimum error of the methods WENO-JS and WENO-Z divided by the error of WENO-DS (rounded to 2 decimal points). We see that for all listed discretizations WENO-DS provides smaller errors, although it was trained only with the fixed spatial discretization $I = 100$.

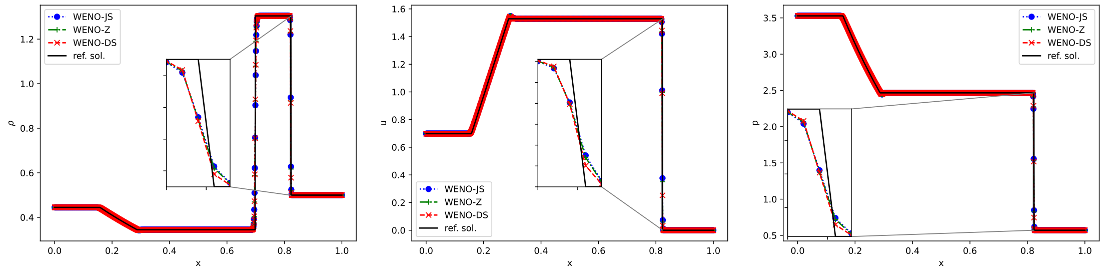


Figure 6.3: Solution of the Lax problem (6.18), using the WENO-JS, WENO-Z and WENO-DS methods, $T = 0.13$, $I = 1000$.

I	100				200				1000			
	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
ρ	0.020890	0.018790	**0.018331**	1.03	0.011144	0.009843	**0.009602**	1.03	0.002791	0.002455	**0.002405**	1.02
u	0.019382	0.018326	**0.017152**	1.04	0.009580	0.008954	**0.008311**	1.05	0.001940	0.001886	**0.001758**	1.04
p	0.026347	0.025225	**0.024275**	1.07	0.012327	0.011640	**0.011098**	1.08	0.002584	0.002531	**0.002435**	1.07

Table 6.1: Comparison of L_1 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Euler equations of gas dynamics for the Lax problem (6.18) for different spatial discretizations, $T = 0.13$.

Figure 6.4 shows the solution of the Sod problem (6.19). Here we computed the solution up to a final time $T = 0.2$.

Moreover, let us present results for some randomly generated initial conditions

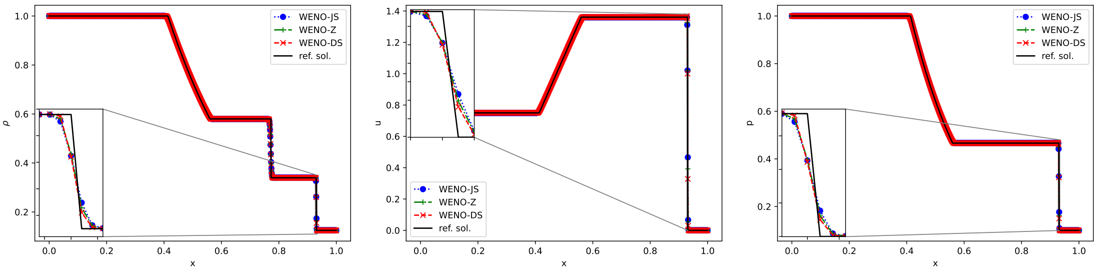


Figure 6.4: Solution of the Sod problem (6.19), using the WENO-JS, WENO-Z and WENO-DS methods, $T = 0.2$, $I = 1000$.

according to Algorithm 3. We consider the following two scenarios:

$$(\rho, u, p) = \begin{cases} (0.5526, 0.9375, 3.9076) & 0 \leq x < 0.5, \\ (0.5665, 0, 0.5430) & 0.5 \leq x \leq 1, \end{cases} \tag{6.20}$$

and

$$(\rho, u, p) = \begin{cases} (1.6282, 0.5117, 1) & 0 \leq x < 0.5, \\ (0.1859, 0, 0.1) & 0.5 \leq x \leq 1, \end{cases} \tag{6.21}$$

with $T = 0.1$. We present the corresponding solutions in Figures 6.5 and 6.6 and the error values can be found in Table 6.2.

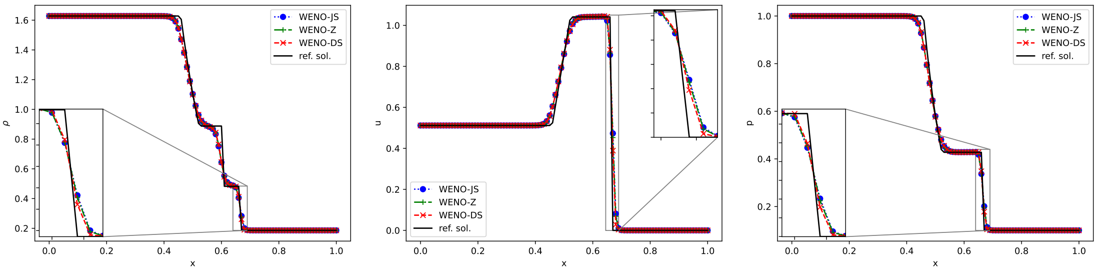


Figure 6.5: Solution of the Euler system with the initial condition (6.20), using the WENO-JS, WENO-Z and WENO-DS methods, $T = 0.1$, $I = 100$.

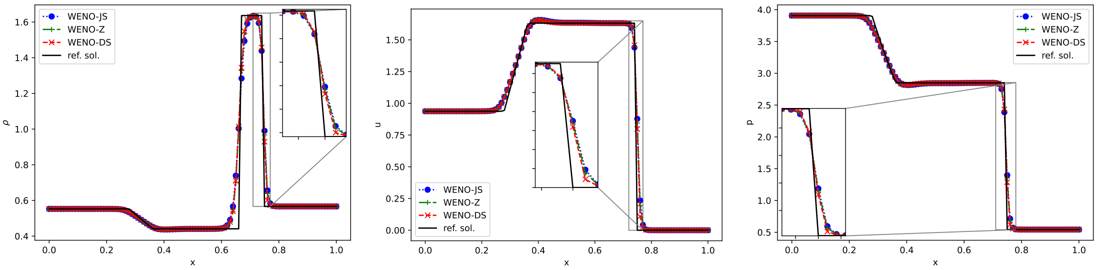


Figure 6.6: Solution of the Euler system with the initial condition (6.21), using the WENO-JS, WENO-Z and WENO-DS methods, $T = 0.1$, $I = 100$.

Finally, we also applied the trained method to the shock entropy wave interaction

	initial condition (6.20)				initial condition (6.21)			
	WENO-JS	WENO-Z	WENO-DS	ratio	WENO-JS	WENO-Z	WENO-DS	ratio
ρ	0.013445	**0.012674**	0.012468	1.02	0.025157	0.022500	**0.021369**	1.05
u	0.011761	0.011190	**0.009919**	1.05	0.021461	0.020004	**0.018047**	1.09
p	0.006893	0.006547	**0.006260**	1.13	0.027735	0.025820	**0.023582**	1.11

Table 6.2: Comparison of L_1 errors of WENO-JS, WENO-Z and WENO-DS methods for the solution of the Euler equations of gas dynamics for the initial conditions (6.20) and (6.21), $T = 0.1$, $I = 100$.

problem [101] with an initial condition

$$(\rho, u, p) = \begin{cases} (3.857143, 2.629369, 10.33333) & -5 \leq x < -4, \\ (1 + 0.2\sin(5x), 0, 1) & -4 \leq x \leq 5. \end{cases} \tag{6.22}$$

In Figure 6.7 we show the solution for ρ, u and p, when the computational domain is divided into 512 uniform cells. We compute the solution for the final time $T = 1.8$. As a reference solution we use the solution computed with WENO-Z method with 2048 space points. This is an example, which has a moving Mach $= 3$ shock interacting with sine waves in density, so the numerical method needs to deal with the physical oscillations contained in a flow. Finally, let us note, that although we have not trained the presented WENO-DS scheme on the parameters that would lead to such a solution, the method is robust enough and can detect the shocks present in the solution very well.

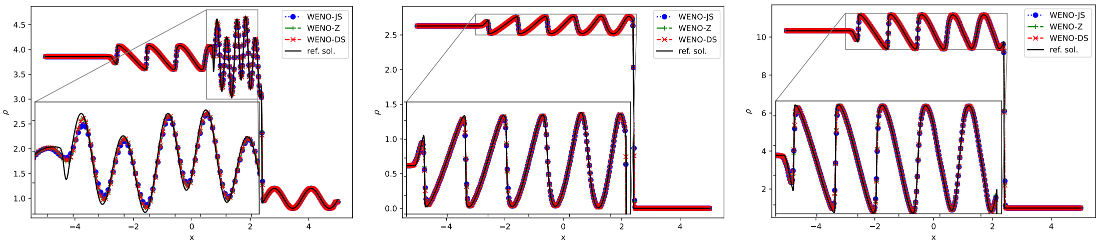


Figure 6.7: Solution of the shock entropy wave interaction problem (6.22), $T = 1.8$, $I = 512$.

Although WENO-DS captures shocks very well and, according to Table 6.1, provides better numerical solutions, the improvements over the base method are rather moderate. However, in the next part of this chapter we will demonstrate the significant superiority of WENO-DS over the standard WENO scheme on the challenging two-dimensional Euler equations of gas dynamics by examining several examples from the literature involving different contact discontinuities, shocks and rarefaction waves.

6.2 Two-dimensional Euler system

In this section we extend the system (6.1) to two dimensions:

$$U_t + F(U)_x + G(U)_y = 0, \tag{6.23}$$

with

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}, \qquad F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{pmatrix}, \qquad G(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{pmatrix} \tag{6.24}$$

for polytropic gas. Here, the variable ρ is the density, u the x-velocity component, v the y-velocity component, E the total energy and p the pressure. Further, it holds

$$p = (\gamma - 1)\Big[E - \frac{\rho}{2}(u^2 + v^2)\Big], \tag{6.25}$$

where γ denotes the ratio of the specific heats and we will use $\gamma \in (1.1, 1.67)$ in our application.

We consider the spatial domain $[0,1] \times [0,1]$ and solve the Riemann problem with the following initial condition

$$(\rho, u, v, p) = \begin{cases} (\rho_1, u_1, v_1, p_1) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (\rho_2, u_2, v_2, p_2) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (\rho_3, u_3, v_3, p_3) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (\rho_4, u_4, v_4, p_4) & x > 0.5 \quad \text{and} \quad y < 0.5, \end{cases} \tag{6.26}$$

and Dirichlet boundary conditions.

The combination of four elementary planar waves is used to define the classification of the Riemann problem. A detailed study of these configurations has been done in [16, 17, 64, 94, 95, 117] and there are 19 different possible configurations for polytropic gas. These are defined by three types of elementary waves, namely a backward rarefaction wave $\overleftarrow{R}$, a backward shock wave $\overleftarrow{S}$, a forward rarefaction wave $\overrightarrow{R}$, a forward shock wave $\overrightarrow{S}$ and a contact discontinuity $J^{\pm}$, where the superscript ± refers to negative and positive contacts.

To obtain the WENO approximations in the two-dimensional example, we apply the procedure described in Section 5.5 using the dimension-by-dimension principle. Thus, we obtain the flux approximations for (6.23) as

$$\begin{aligned} \frac{1}{\Delta x}\big(\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}\big) &= \big(F(U)\big)_x\big|_{(x_i,y_j)} + O\big(\Delta x^5\big), \\ \frac{1}{\Delta y}\big(\hat{k}_{i,j+\frac{1}{2}} - \hat{k}_{i,j-\frac{1}{2}}\big) &= \big(G(U)\big)_y\big|_{(x_i,y_j)} + O\big(\Delta y^5\big), \end{aligned} \tag{6.27}$$

with the uniform grid defined by the nodes (x_i, y_j), $\Delta x = x_{i+1} - x_i$, $\Delta y = y_{j+1} - y_j$, $i = 0, \dots, I$, $j = 0, \dots, J$.

For the deep learning part, the improved smoothness indicators (6.5) with (6.6) and (6.7) are used.

6.2.1 Structure of neural network and the training procedure

In our two-dimensional implementation we proceed with the training as described in Section 6.1.1. In our experiments we use the probability $\varphi = 0.5$ and set the maximum number of opened problems to 150. We use again the Adam optimizer, but we change the learning rate to 0.001.

During the training, one more adjustment has to be made as compared to the one-dimensional implementation. As we do not know the exact solutions for implemented two-dimensional examples, we have to create a data set. For this purpose we compute the reference solutions using the WENO-Z method on a fine grid of $I \times J = 400 \times 400$ space points up to the given final time T, where t_n represents the time points, $n = 0, \dots, N$. More details on the construction of the reference solutions will be given further in this section. During training, we compute the numerical solutions on a grid of $I \times J = 100 \times 100$ space points. At the beginning of a training we randomly select a problem from a data set and perform a single time step to get to the time t_{n+1}, using CNN to predict the multipliers δ_m. However, by performing a single time step on a coarse grid, we do not match the time step size of the fine precomputed solutions, as the adaptive time step size is used. So we simply take the closest reference solution from the data set, use it as an initial condition, and do another small time step to get a reference solution at the time instance t_{n+1}. Then we compute the loss and its gradient with respect to the weights of the CNN. The rest of the training procedure remains the same as described in Section 6.1.1.

Loss function

Loss function and validation metrics are used as defined in (6.10) and (6.11).

The loss function from (6.10) for training is adapted such that

$$\begin{aligned} LOSS_{\mathrm{MSE}}(\rho, u, v, p) = LOSS_{\mathrm{MSE}}(\rho) + LOSS_{\mathrm{MSE}}(u) \\ + LOSS_{\mathrm{MSE}}(v) + LOSS_{\mathrm{MSE}}(p), \end{aligned} \tag{6.28}$$

and for the validation during training we use the adaptation from (6.11)

$$L_1(\rho, u, v, p) = L_1(\rho) + L_1(u) + L_1(v) + L_1(p). \tag{6.29}$$

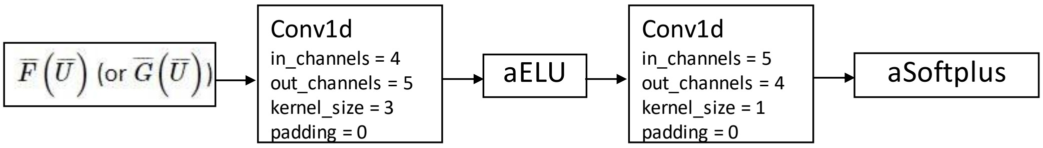
When we plot the error on validation problems, we rescale the values for each validation problem to be located in the interval $[0,1]$ using the relationship

$$L_1^*(\rho,u,v,p) = \frac{L_1^l(\rho,u,v,p)}{\max\limits_{l=0,\dots,L}\left(L_1^l(\rho,u,v,p)\right)}, \qquad l=0,\dots,L, \tag{6.30}$$

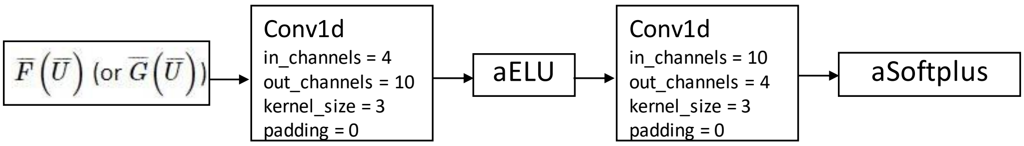
where L denotes the total number of training steps.

Size of the neural network

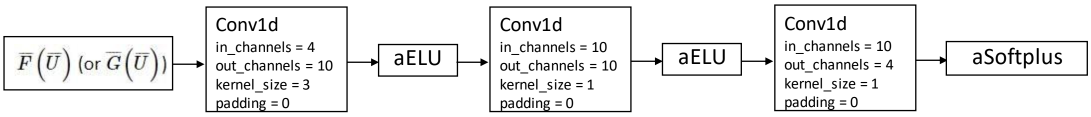
In our implementation, we considered different structures of CNNs and carried out numerous experiments with them. First, we used a rather simple CNN with only two layers and a receptive field of width 3. The structure is shown in Figure 6.8a. The advantage of this setting is its computational efficiency. Second, we used a CNN with the same number of layers, but we increased the number of channels and made the receptive field wider. The structure is shown in Figure 6.8b. Finally, we used only a receptive field of width 3, but added one more layer and used a more complex CNN, as shown in Figure 6.8c. Each of these CNNs gave interesting results and we summarize them in Section 6.2.2.



(a) Two hidden layers, lower number of channels, receptive field of size 3.



(b) Two hidden layers, higher number of channels, receptive field of size 5.



(c) Three hidden layers, higher number of channels, receptive field of size 3.

Figure 6.8: Different structures of the CNN used for the training of WENO-DS.

As it can be seen, we have 4 input channels in the first hidden layer and 4 output channels in the last hidden layer in each CNN. These represent the dimension of the solution U from (6.24). In this way, the CNN also receives information from other variables, which can be useful for improving the numerical solution. The input $\bar{F}(\bar{U})$, respectively $\bar{G}(\bar{U})$ represents the numerical approximation after the projection using left eigenvectors and after applying the flux splitting method.

Construction of the data set for the CNN training procedure

For each of the 19 configurations of the Riemann problem (6.26), the specific relations must be satisfied by the initial data and the symmetry properties of the solution. We present the formulas given in [95] and create the data sets for the CNN training according to these formulas.

We define

$$\Phi_{lr} := \frac{2\sqrt{\gamma}}{\gamma-1}\Big(\sqrt{\frac{p_l}{\rho_l}} - \sqrt{\frac{p_r}{\rho_r}}\Big), \quad \Psi_{lr}^2 := \frac{(p_l-p_r)(\rho_l-\rho_r)}{\rho_l\rho_r}, \quad (\Psi_{lr} > 0), \tag{6.31}$$

and

$$\Pi_{lr} := \Big(\frac{p_l}{p_r} + \frac{(\gamma-1)}{(\gamma+1)}\Big)\Big/\Big(1 + \frac{(\gamma-1)}{(\gamma+1)}\frac{p_l}{p_r}\Big). \tag{6.32}$$

In the numerical examples presented in Section 6.2.2 we list more specific relations for each of the given examples that are sufficient to uniquely define the solution. Following these relations, we randomly generate the initial data and construct our data sets.

6.2.2 Numerical examples

To demonstrate the efficiency of the proposed method, we present here the numerical results obtained with the WENO-DS method after the CNN training procedure. Note that the CNN training procedure has to be performed only once as offline training for each of the three examples: configuration 2, configuration 3 and configuration 16. No additional training was performed for the next examples: configuration 11 and configuration 19, since we show the results using the same trained CNN from the previous examples. Finally, in the last part of this chapter we perform two more trainings with a larger CNN and illustrate the results. More details can be found in the corresponding sections. For the implementation we use Python with the deep learning library PyTorch.

For the system of ODEs (6.15) we use a third-order TVD Runge-Kutta method (5.59), where we evaluate the CNN during training, as well as during testing, in each Runge-Kutta stage.

Let us note, that for all trainings we used fixed spatial discretization $I \times J = 100 \times 100$ space points. When we test the method and compare the computational costs, we consider also the spatial discretization 50×50 and 200×200 space points.

For the temporal discretization we use an adaptive step size

$$\Delta t = 0.6 \min\Big(\frac{\Delta x}{a}, \frac{\Delta y}{a}\Big), \tag{6.33}$$

with

$$a = \max_{\substack{i=0,\dots,I\\ j=0,\dots,J}} (|\lambda^{+}_{i,j}|, |\lambda^{-}_{i,j}|) \quad \lambda^{\pm} = V \pm c, \quad V = \sqrt{u^2+v^2} \quad c^2 = \gamma \frac{p}{\rho}, \tag{6.34}$$

where u, v are the velocities and c is the local speed of sound.

In the sequel we enumerate the different configurations of initial conditions according to [64].

Configuration 2

This is the configuration with four rarefaction waves: $\overrightarrow{R}_{21}$, $\overleftarrow{R}_{32}$, $\overleftarrow{R}_{34}$, $\overrightarrow{R}_{41}$. The detailed analysis was done in [95, 117] and we have to satisfy the following relations for this case:

$$\begin{aligned} &u_2 - u_1 = \Phi_{21}, \quad u_4 - u_3 = \Phi_{34}, \quad u_3 = u_2, \quad u_4 = u_1, \\ &v_4 - v_1 = \Phi_{41}, \quad v_2 - v_3 = \Phi_{32}, \quad v_2 = v_1, \quad v_3 = v_4, \end{aligned} \tag{6.35}$$

with the compatibility conditions $\Phi_{21} = -\Phi_{34}$ and $\Phi_{41} = -\Phi_{32}$. Moreover, for a polytropic gas the equations

$$\rho_l/\rho_r = (p_l/p_r)^{1/\gamma} \quad \text{for} \quad (l,r) \in \{(2,1),(3,4),(3,2),(4,1)\}, \tag{6.36}$$

have to be included. Furthermore, we have $\rho_2 = \rho_4$, $\rho_1 = \rho_3$, $p_1 = p_3$, $p_2 = p_4$, $u_2 - u_1 = v_4 - v_1$ and $u_4 - u_3 = v_2 - v_3$.

We use for creating of the data set the values

$$\begin{gathered} \rho_1 \in \mathcal{U}[0.7, 2], \quad \rho_2 \in \mathcal{U}[0.5, \rho_1], \quad p_1 \in \mathcal{U}[0.2, 1.5], \\ u_1 \in \mathcal{U}[-1, 1], \quad v_1 = u_1, \quad \gamma \in (1.1, 1.67), \end{gathered} \tag{6.37}$$

and for the other values we use the relations (6.35), (6.36) with (6.31). We also compute the reference solutions using the WENO-Z method on a grid $I \times J = 400 \times 400$ space points up to the final time $T \in \mathcal{U}[0.1, 0.2]$ and create the data set consisting of 50 reference solutions.

For training, we use the training procedure described in Section 6.2.1. First, we use the simplest CNN structure shown in Figure 5.3a and perform the training for the total number of 4000 training steps. We plot the evolution of the L_1^* error (6.30) for the validation problems in Figure 6.9. Note that these problems were not included in the training data, and the initial conditions of these problems were generated analogously to the construction of the training data set. For these problems, we measured the error every 100 training steps and at a randomly chosen final time T. We select the final model based on the evolution of the error of the validation set. We see that the error decreases up to a certain point for all problems and then starts to increase for some problems. Longer training would lead to overfitting of the training data. Finally, we choose the final model from the training step 2800 and present the results using this model. This is a training step, in which the sum of the validation values (6.29) over all validation problems obtains its lowest value.

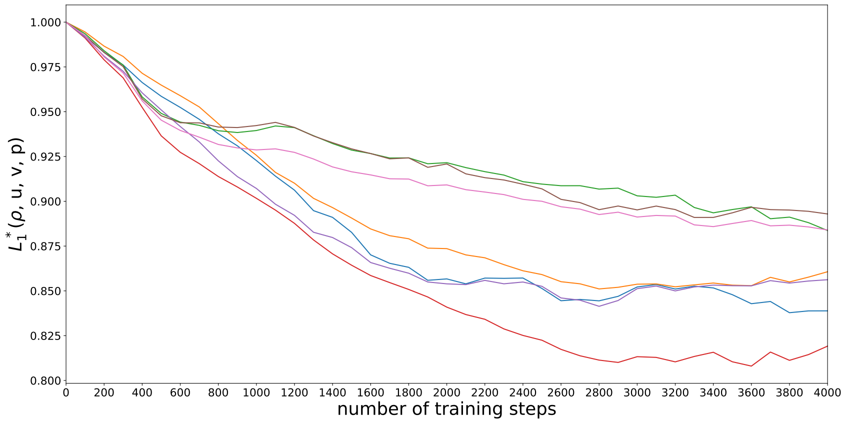


Figure 6.9: Training evolution corresponding to configuration 2: The values of the rescaled validation metrics (6.30) for different validation problems evaluated each 100 training steps.

As a test problem we use the problem from [64] with $\gamma = 1.4$, $T = 0.2$ and the initial condition

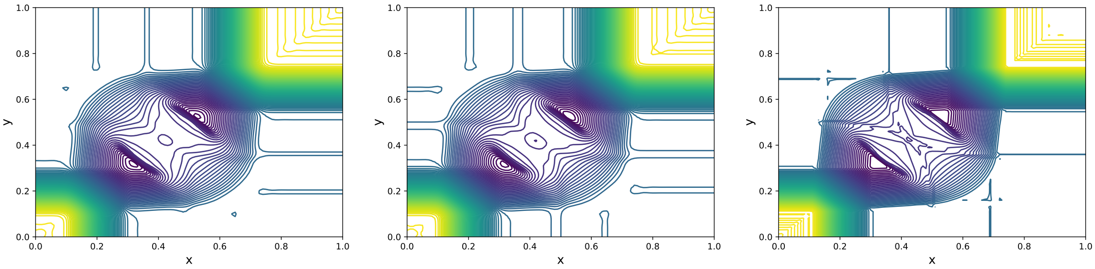
$$(\rho, u, v, p) = \begin{cases} (1, 0, 0, 1) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (0.5197, -0.7259, 0, 0.4) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (1, -0.7259, -0.7259, 1) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (0.5197, 0, -0.7259, 0.4) & x > 0.5 \quad \text{and} \quad y < 0.5. \end{cases} \tag{6.38}$$

The results are shown in Table 6.3. As can be seen, we achieve a significant error improvement for all four variables and for different discretizations. It should be noted that we trained only with the discretization of 100×100 space points and did not retrain the CNN for different discretizations. We refer to the error of the WENO-Z method divided by the error of WENO-DS (rounded to 2 decimal points) as the 'ratio'. The density contour plots are shown in Figure 6.10 and the absolute pointwise errors for the density are shown in Figure 6.11. Moreover, we show in Figure 6.12 the density contour plots for the spatial resolution $I \times J = 400 \times 400$, i.e. the resolution of the reference data. As it can be seen, at the susceptibility area, WENO-Z produces noticeable oscillations, which is not a case for WENO-DS.

Finally, we want to compare the computational cost of WENO-DS compared to the WENO-Z scheme in solving the test problem shown in Figure 6.13. Using a logarithmic scale, we plot the computation time against the L_1 error averaged over the four variables ρ, u, v, p. We conclude, that compared to WENO-Z method, WENO-DS is computationally more effective. As can be seen, WENO-DS has a slightly longer computation time. This is due to the evaluation of the CNN. It is important to note that the structure of the CNN should be as small as possible. Using a larger CNN would shift the red line more to the right and the WENO-DS

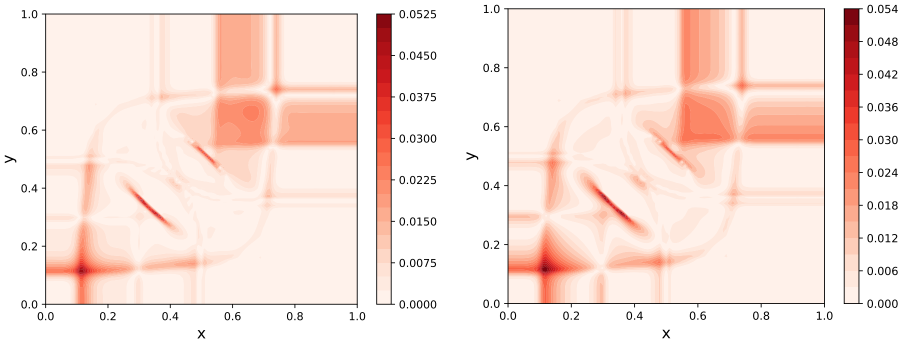
$I \times J$	50×50			100×100			200×200		
	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	0.012488	0.010722	1.16	0.005465	0.004648	1.18	0.001862	0.001547	1.20
u	0.014363	0.011986	1.20	0.006153	0.005066	1.21	0.002053	0.001627	1.26
v	0.014363	0.011986	1.20	0.006153	0.005066	1.21	0.002053	0.001627	1.26
p	0.013113	0.011510	1.14	0.005619	0.004899	1.15	0.001879	0.001587	1.18

Table 6.3: Comparison of L_1 errors of WENO-Z and WENO-DS methods for the solution of the Euler system with the initial condition (6.38) for different spatial discretizations, $T = 0.2$.



(a) WENO-DS (b) WENO-Z (c) reference solution

Figure 6.10: Density contour plot for the solution of the Riemann problem with the initial condition (6.38), $I \times J = 100 \times 100$, $T = 0.2$.

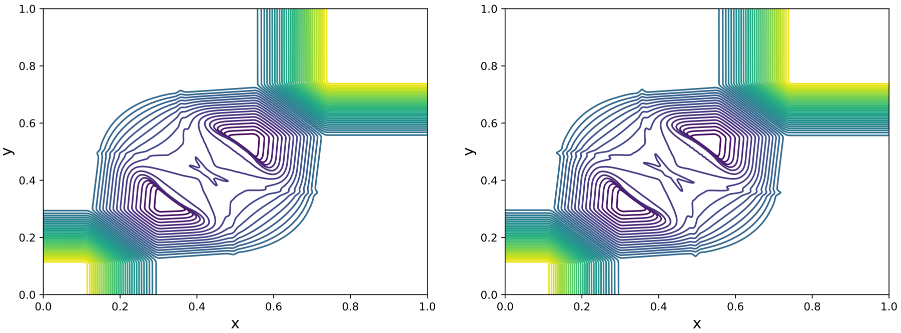


(a) WENO-DS (b) WENO-Z

Figure 6.11: Absolute pointwise errors for the density solution of the Riemann problem with the initial condition (6.38), $I \times J = 100 \times 100$, $T = 0.2$.

would be computationally inefficient. Efficiency would only be maintained if we could get even smaller errors with a larger CNN.

It should be noted that if we were to test the method on another unseen test problems using the initial data from the previously described range, we would obtain very similar error improvements in those cases.



(a) WENO-DS (b) WENO-Z

Figure 6.12: Density contour plot for the solution of the Riemann problem with the initial condition (6.38), $I \times J = 400 \times 400$, $T = 0.2$.

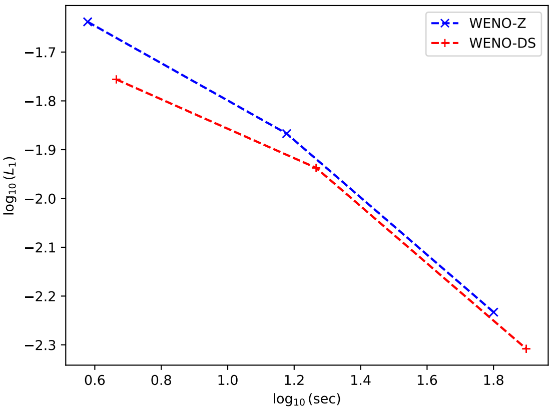


Figure 6.13: Comparison of computational cost against L_1 error of the solution of the Riemann problem with the initial condition (6.38).

Configuration 3

This is the configuration with four shock waves: $\overleftarrow{S}_{21}$, $\overleftarrow{S}_{32}$, $\overleftarrow{S}_{34}$, $\overleftarrow{S}_{41}$. According to [95], in this case we have the following equations that must be satisfied:

$$\begin{aligned} &u_2 - u_1 = \Psi_{21}, \quad u_3 - u_4 = \Psi_{34}, \quad u_3 = u_2, \quad u_4 = u_1, \\ &v_4 - v_1 = \Psi_{41}, \quad v_3 - v_2 = \Psi_{32}, \quad v_2 = v_1, \quad v_3 = v_4, \end{aligned} \tag{6.39}$$

and for polytropic gas the equations

$$\rho_l/\rho_r = \Pi_{lr} \quad \text{for} \quad (l,r) \in \{(2,1),(3,4),(3,2),(4,1)\} \tag{6.40}$$

are added. This gives the compatibility conditions $\Psi_{21} = \Psi_{34}$ and $\Psi_{41} = \Psi_{32}$. Furthermore, we have $\rho_2 = \rho_4$, $p_2 = p_4$ and $u_2 - u_1 = v_4 - v_1$.

In this case, we use for creating the dataset the values

$$\begin{gathered}\rho_1 \in \mathcal{U}[1,2], \quad \rho_2 \in \mathcal{U}[0.5,1], \quad p_1 \in \mathcal{U}[1,2], \\ u_1 \in \mathcal{U}[-0.25,0.25], \quad v_1 = u_1, \quad \gamma \in (1.1, 1.67),\end{gathered} \tag{6.41}$$

and for the other values we use the relations (6.39), (6.40) with (6.31) and (6.32). Similar to the previous example, we compute the reference solutions using the WENO-Z method on a grid with $I \times J = 400 \times 400$ space points up to the final time $T \in \mathcal{U}[0.1, 0.3]$ and create the data set consisting of 50 reference solutions.

We proceed with training as described in the previous section, using the same CNN structure as shown in Figure 5.3a. Again, we train only on the discretization $I \times J = 100 \times 100$ space steps. We run the training for 4000 training steps and plot the evolution of the validation metrics (6.30) for the validation problems in Figure 6.14. We measured the error every 100 training steps and at the randomly chosen final time T. We compute the sum of the validation values (6.29) over all validation problems. Its lowest value is obtained in a training step 3200 so we choose the final model from that training step and present the results for the test problem with $\gamma = 1.4$, $T = 0.3$, and initial condition from [64]

$$(\rho, u, v, p) = \begin{cases} (1.5, 0, 0, 1.5) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (0.5323.1.206, 0, 0.3) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (0.138, 1.206, 1.206, 0.029) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (0.5323, 0, 1.206, 0.3) & x > 0.5 \quad \text{and} \quad y < 0.5. \end{cases} \tag{6.42}$$

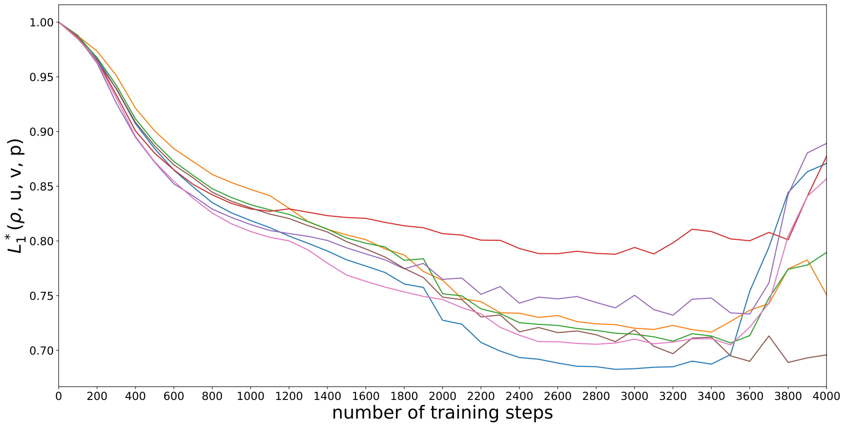
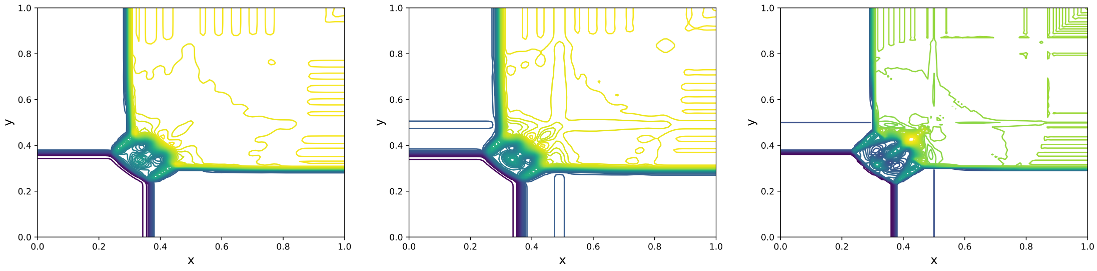


Figure 6.14: Training evolution corresponding to configuration 3: The values of the rescaled validation metrics (6.30) for different validation problems evaluated each 100 training steps.

We compare the results in Table 6.4. As can be seen, we achieve a large error improvement for all discretizations listed. The density contour plots can be found in Figure 6.15 and the absolute pointwise errors for the density in Figure 6.16. Here it can be seen that the error of WENO-DS is significantly lower in the areas of the shock contacts. Moreover, we compare the density contour plots on the resolution of the reference data in Figure 6.17.

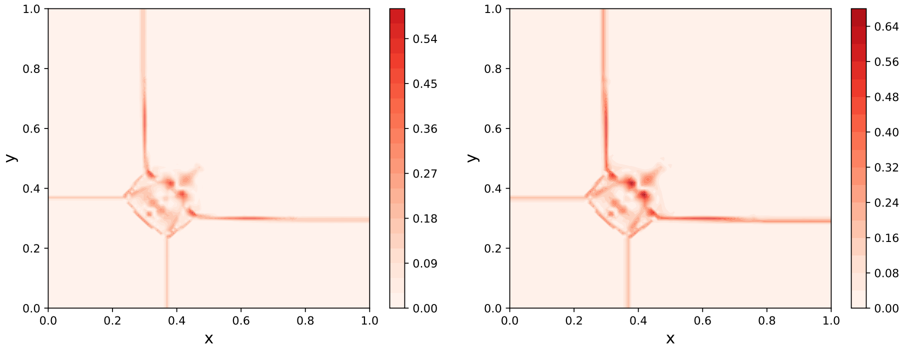
$I \times J$	50 × 50			100 × 100			200 × 200		
	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	0.038682	0.027906	1.39	0.019232	0.012817	1.50	0.007454	0.004657	1.60
u	0.034692	0.027638	1.26	0.019588	0.015043	1.30	0.008249	0.005810	1.42
v	0.034692	0.027638	1.26	0.019588	0.015043	1.30	0.008249	0.005810	1.42
p	0.038920	0.030888	1.26	0.018666	0.014041	1.33	0.007275	0.005001	1.45

Table 6.4: Comparison of L_1 errors of WENO-Z and WENO-DS methods for the solution of the Euler system with the initial condition (6.42) for different spatial discretizations, $T = 0.3$.



(a) WENO-DS (b) WENO-Z (c) reference solution

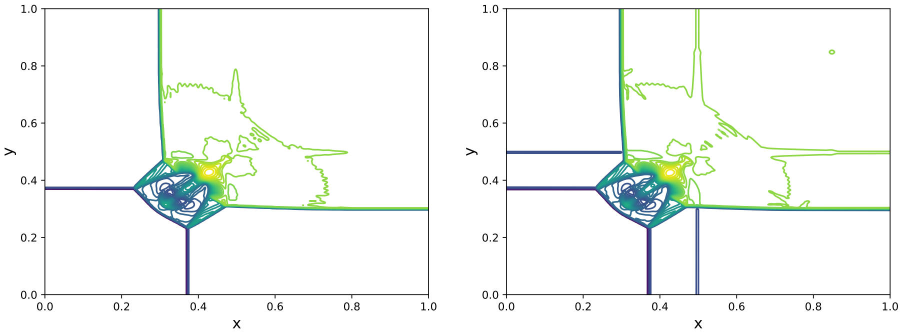
Figure 6.15: Density contour plot for the solution of the Riemann problem with the initial condition (6.42), $I \times J = 100 \times 100$, $T = 0.3$.



(a) WENO-DS (b) WENO-Z

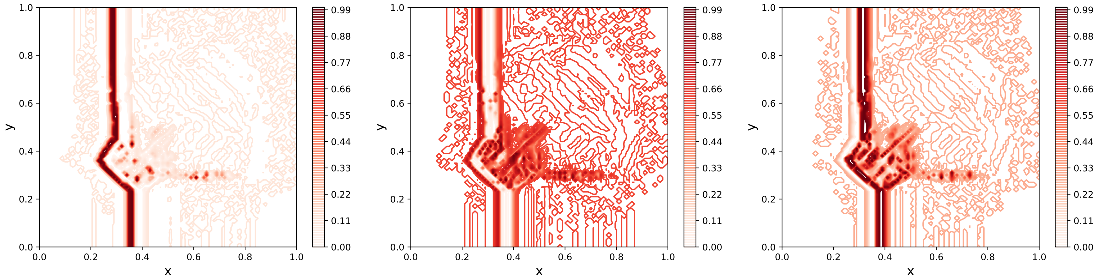
Figure 6.16: Absolute pointwise errors for the density solution of the Riemann problem with the initial condition (6.42), $I \times J = 100 \times 100$, $T = 0.3$.

In Figure 6.18 we also compare the weights ω_m^Z, $m = 0, 1, 2$ (5.32) and the updated weights ω_m^{DS}, $m = 0, 1, 2$ (5.45) with the improved smoothness indicators (6.5). We plot these weights, corresponding to the positive part of a flux from the flux splitting, using WENO-Z and WENO-DS for the previous test problem at the final time $T = 0.3$. Since we apply the principle dimension-by-dimension, we present the weights only for the approximation of the flux $F(U)$. For the approximations of the flux $G(U)$, we could obtain these weights in this example using symmetry. As can be seen, WENO-DS is much better at localizing the shock from the other direction as well, which has a significant impact on error improvement.

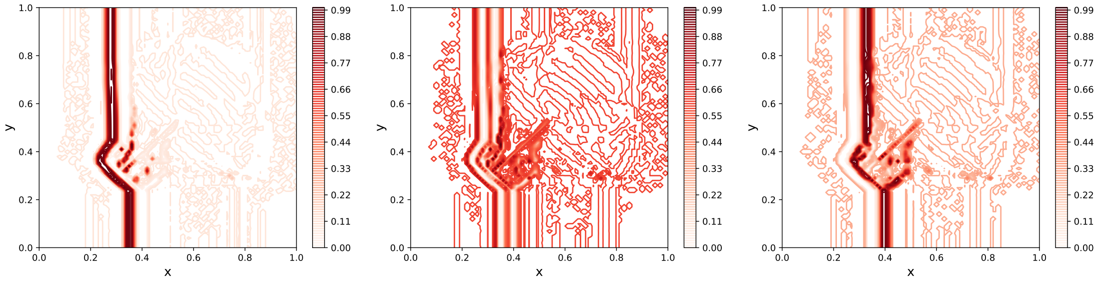


(a) WENO-DS (b) WENO-Z

Figure 6.17: Density contour plot for the solution of the Riemann problem with the initial condition (6.42), $I \times J = 400 \times 400$, $T = 0.3$.



(a) WENO-DS, ω_0^{DS} (b) WENO-DS, ω_1^{DS} (c) WENO-DS, ω_2^{DS}



(d) WENO-Z, ω_0^{Z} (e) WENO-Z, ω_1^{Z} (f) WENO-Z, ω_2^{Z}

Figure 6.18: Configuration 3: Comparison of the nonlinear weights ω_m^{DS} and ω_m^{Z}, $m = 0, 1, 2$ corresponding to the flux $F(U)$, $I \times J = 100 \times 100$, $T = 0.3$.

Finally, let us compare the computational cost of WENO-DS for the previously discussed test problem shown in Figure 6.19. We see that WENO-DS is much more computationally efficient compared to WENO-Z. Again, if we test the method on the unseen problems, but with the same initial configuration, we would get analogous significant error improvements.

Configuration 16

This is the configuration with the combination of rarefaction wave, shock wave and contact discontinuities: $\overleftarrow{R}_{21}$, J_{32}^{-}, J_{34}^{+}, $\overrightarrow{S}_{41}$. As shown in [95], the following relations

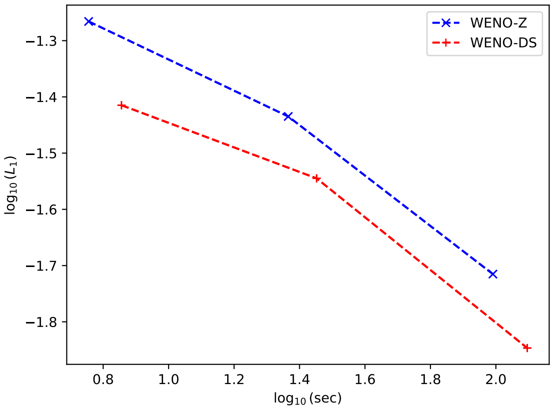


Figure 6.19: Comparison of computational cost against L_1 error of the solution of the Riemann problem with the initial condition (6.42).

must hold for this case

$$\begin{aligned} u_1 - u_2 = \Phi_{21}, \quad u_3 = u_4 = u_1, \\ v_4 - v_1 = \Psi_{41}, \quad v_3 = v_2 = v_1, \quad p_1 < p_2 = p_3 = p_4, \end{aligned} \tag{6.43}$$

and for polytropic gas we add the equation (6.36) for a rarefaction and (6.40) for a shock wave between the lth and rth quadrants.

For our data set we use the values

$$\begin{aligned} \rho_4 \in \mathcal{U}[1,2], \quad \rho_3 \in \mathcal{U}[0.5,\rho_4], \quad p_1 \in \mathcal{U}[0.3,1], \quad p_2 \in \mathcal{U}[1,1.5], \\ u_1 \in \mathcal{U}[-0.25,0.25], \quad v_1 = u_1, \quad \gamma \in (1.1,1.67). \end{aligned} \tag{6.44}$$

To compute the data set consisting of 50 reference solutions, we use the WENO-Z method on a grid with $I \times J = 400 \times 400$ space points up to the final time $T \in \mathcal{U}[0.1,0.2]$.

We train the CNN with the structure shown in Figure 5.3a as in the previous examples on the discretization with $I \times J = 100 \times 100$ space steps for the total number of 2000 training steps. We show the evolution of the validation metrics (6.30) in Figure 6.20 and choose the model from training step 1900 as final WENO-DS scheme.

We test the trained WENO-DS on a test problem [64] with $\gamma = 1.4$, $T = 0.2$ and the initial condition

$$(\rho,u,v,p) = \begin{cases} (0.5313,0.1,0.1,0.4) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (1.0222,-0.6179,0.1,1) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (0.8,0.1,0.1,1) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (1,0.1,0.8276,1) & x > 0.5 \quad \text{and} \quad y < 0.5. \end{cases} \tag{6.45}$$

We compare the results in Table 6.5 and the density contour plots can be found in

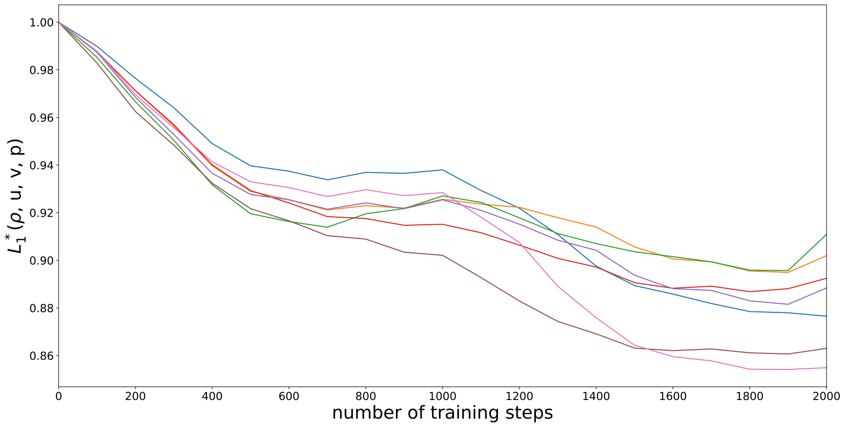


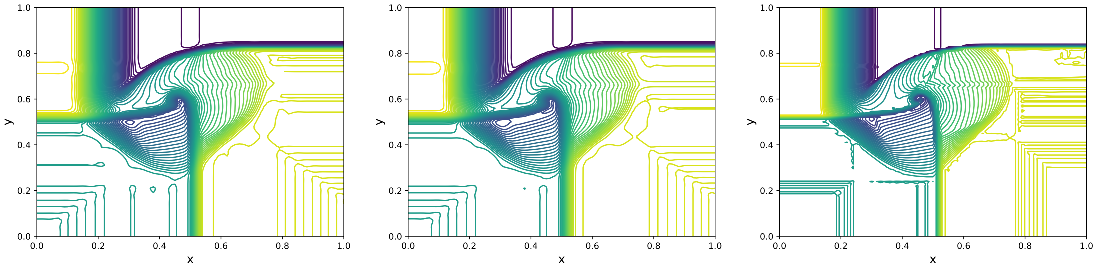
Figure 6.20: Training evolution corresponding to configuration 16: The values of the rescaled validation metrics (6.30) for different validation problems evaluated each 100 training steps.

Figure 6.21. As can be seen, WENO-DS outperforms WENO-Z and has smaller L_1 errors in all cases. In addition, we plot the absolute pointwise errors for the density solution and show them in Figure 6.22. The density contour plots for the spatial resolution $I \times J = 400 \times 400$ can be found in Figure 6.23.

For another unseen test problems with the same initial configurations, we would again obtain analogous significant error improvements.

$I \times J$	50×50			100×100			200×200		
	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	0.010980	0.009877	1.11	0.004834	0.004327	1.12	0.001827	0.001624	1.12
u	0.012464	0.011287	1.10	0.005989	0.005326	1.12	0.002223	0.001913	1.16
v	0.015020	0.013932	1.08	0.006609	0.006172	1.07	0.002527	0.002298	1.10
p	0.010594	0.009644	1.10	0.004236	0.003820	1.11	0.001576	0.001392	1.13

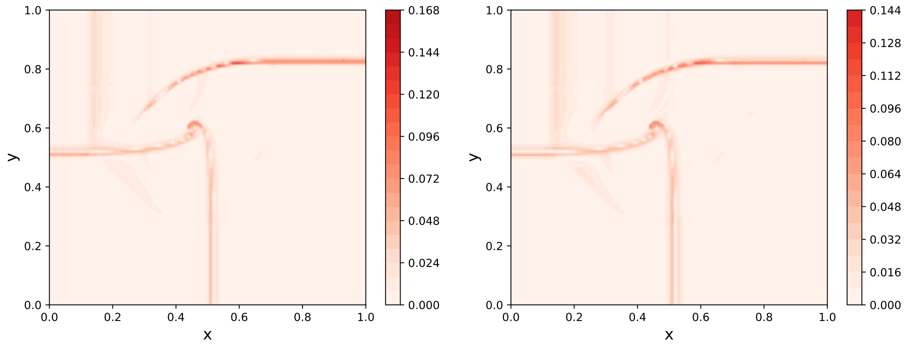
Table 6.5: Comparison of L_1 errors of WENO-Z and WENO-DS methods for the solution of the Euler system with the initial condition (6.45) for different spatial discretizations, $T = 0.2$.



(a) WENO-DS (b) WENO-Z (c) reference solution

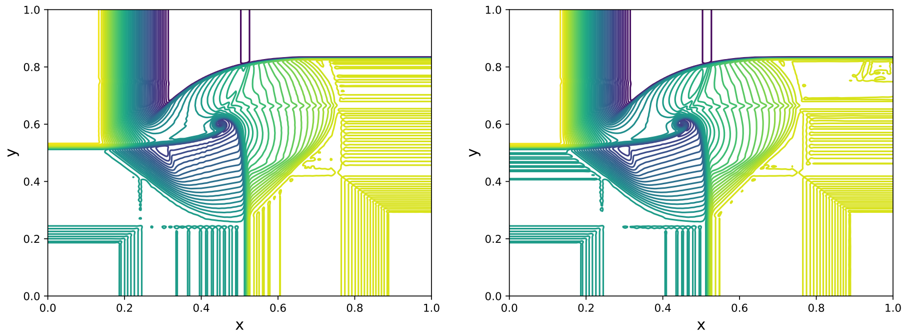
Figure 6.21: Density contour plot for the solution of the Riemann problem with the initial condition (6.45), $I \times J = 100 \times 100$, $T = 0.2$.

We also compare the weights ω_m^Z, $m = 0, 1, 2$ and the updated weights ω_m^{DS}, $m = 0, 1, 2$. As the solution is not symmetric, we compare the weights corresponding



(a) WENO-DS (b) WENO-Z

Figure 6.22: Absolute pointwise errors for density solution of the Riemann problem with the initial condition (6.45), $I \times J = 100 \times 100$, $T = 0.2$.



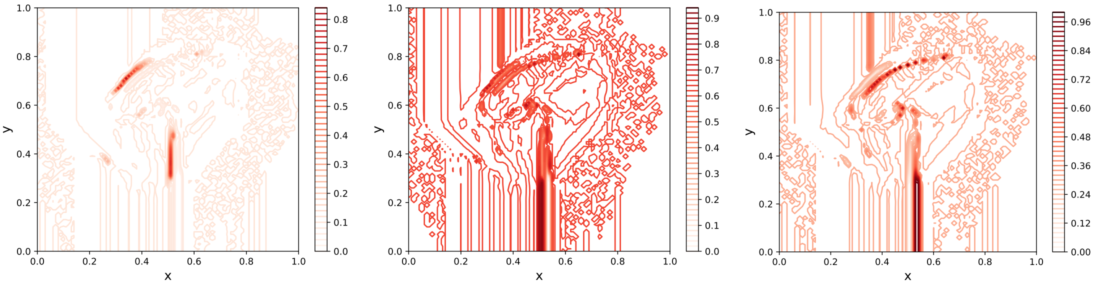
(a) WENO-DS (b) WENO-Z

Figure 6.23: Density contour plot for the solution of the Riemann problem with the initial condition (6.45), $I \times J = 400 \times 400$, $T = 0.2$.

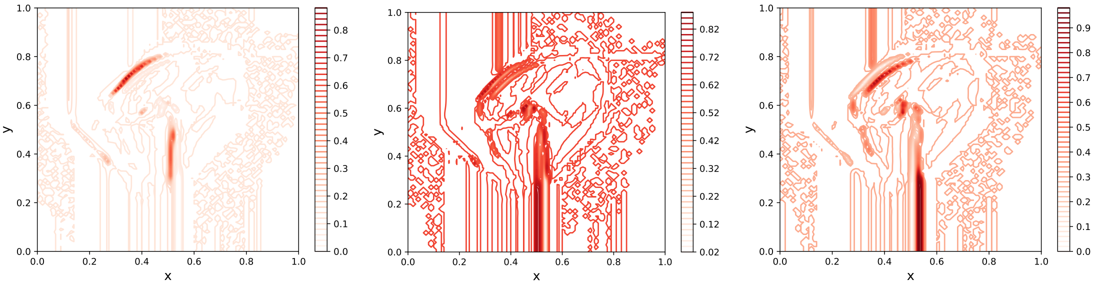
to the flux $F(U)$ in Figure 6.24 and the weights corresponding to the flux $G(U)$ in Figure 6.25. In both cases, we plot these weights, corresponding to the positive part of a flux from the flux splitting, for the previous test problem at the final time $T = 0.2$.

Configuration 11 and configuration 19

In the previous sections, we trained three WENO-DS methods for three different types of configurations. We denote by WENO-DS (C2), WENO-DS (C3), and WENO-DS (C16) the methods according to the configurations, on which the methods were trained. In this section, we test these methods on the unseen problems containing the combination of rarefaction waves, shock waves, and contact discontinuities. First, we consider configuration 11 ($\overleftarrow{S}_{21}$, J^+_{32}, J^+_{34}, $\overleftarrow{S}_{41}$) with the test

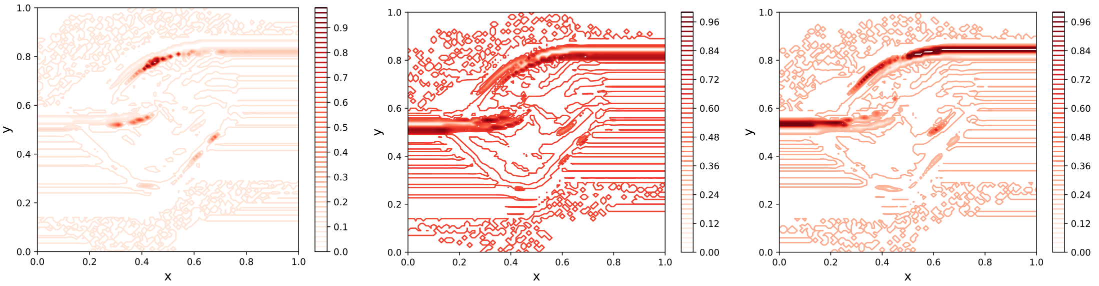


(a) WENO-DS, ω_0^{DS} (b) WENO-DS, ω_1^{DS} (c) WENO-DS, ω_2^{DS}

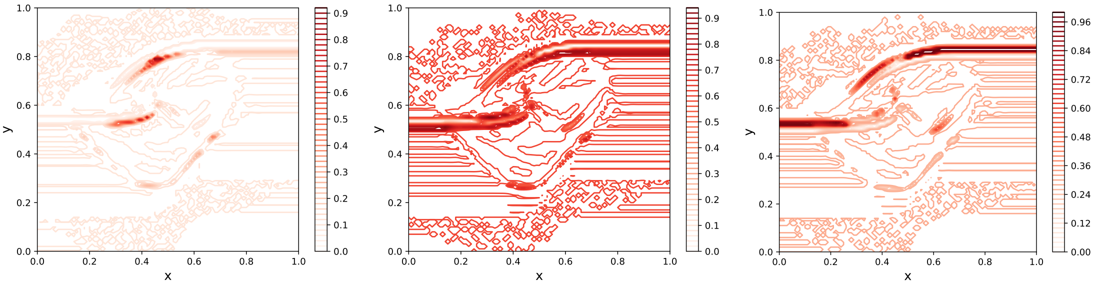


(d) WENO-Z, ω_0^Z (e) WENO-Z, ω_1^Z (f) WENO-Z, ω_2^Z

Figure 6.24: Configuration 16: Comparison of the nonlinear weights ω_m^{DS} and ω_m^Z, $m = 0, 1, 2$ corresponding to the flux $F(U)$, $I \times J = 100 \times 100$, $T = 0.2$.



(a) WENO-DS, ω_0^{DS} (b) WENO-DS, ω_1^{DS} (c) WENO-DS, ω_2^{DS}



(d) WENO-Z, ω_0^Z (e) WENO-Z, ω_1^Z (f) WENO-Z, ω_2^Z

Figure 6.25: Configuration 16: Comparison of the nonlinear weights ω_m^{DS} and ω_m^Z, $m = 0, 1, 2$ corresponding to the flux $G(U)$, $I \times J = 100 \times 100$, $T = 0.2$.

problem with $\gamma = 1.4$, $T = 0.3$, and the initial condition from [64] given by

$$(\rho, u, v, p) = \begin{cases} (1, 0.1, 0, 1) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (0.5313, 0.8276, 0, 0.4) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (0.8, 0.1, 0, 0.4) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (0.5313, 0.1, 0.7276, 0.4) & x > 0.5 \quad \text{and} \quad y < 0.5. \end{cases} \tag{6.46}$$

Second, we test the models on the configuration 19 (J_{21}^+, $\overleftarrow{S}_{32}$, J_{34}^-, $\overrightarrow{R}_{41}$) with the test problem with $\gamma = 1.4$, $T = 0.3$ and the initial condition from [64] given by

$$(\rho, u, v, p) = \begin{cases} (1, 0, 0.3, 1) & x > 0.5 \quad \text{and} \quad y > 0.5, \\ (2, 0, -0.3, 1) & x < 0.5 \quad \text{and} \quad y > 0.5, \\ (1.0625, 0, 0.2145, 0.4) & x < 0.5 \quad \text{and} \quad y < 0.5, \\ (0.5197, 0, -0.4259, 0.4) & x > 0.5 \quad \text{and} \quad y < 0.5. \end{cases} \tag{6.47}$$

We summarize the results in Tables 6.6 and 6.7. We use boldface to indicate the best performing WENO-DS scheme. As can be seen, the method trained on problems containing only rarefaction waves has the worst ability to generalize to unseen problems. On the other hand, by using methods trained on problems containing shock waves or a combination of contact discontinuities, rarefaction, and shock waves, we obtain the error improvements even on unseen problems with different initial configurations. We would like to emphasize that the test problems in this section are far from the problems included in the training and validation sets. This is not only due to the choice of initial data, but also to the combination of rarefaction, shock waves and their direction, and positive and negative contact discontinuities.

	Configuration 11						
	WENO-Z	WENO-DS (C2)	ratio	WENO-DS (C3)	ratio	WENO-DS (C16)	ratio
ρ	0.007792	0.008000	0.97	**0.006783**	**1.15**	0.007538	1.03
u	0.008003	0.008701	0.92	0.007846	1.02	**0.007840**	**1.02**
v	0.007692	0.008300	0.93	**0.007161**	**1.07**	0.007370	1.04
p	0.005883	0.006467	0.91	**0.005115**	**1.15**	0.005776	1.02

Table 6.6: Comparison of L_1 errors of WENO-Z and WENO-DS methods trained on different configurations (C2, C3 and C16) for the solution of the Euler system with the initial condition (6.46), $I \times J = 100 \times 100$, $T = 0.3$.

	Configuration 19						
	WENO-Z	WENO-DS (C2)	ratio	WENO-DS (C3)	ratio	WENO-DS (C16)	ratio
ρ	0.014844	0.014463	1.03	**0.013702**	**1.08**	0.013841	1.07
u	0.003749	**0.003562**	**1.05**	0.003689	1.02	0.003574	1.05
v	0.009891	0.009502	1.04	0.009791	1.01	**0.009245**	**1.07**
p	0.006123	0.005922	1.03	**0.005595**	**1.09**	0.005844	1.05

Table 6.7: Comparison of L_1 errors of WENO-Z and WENO-DS methods trained on different configurations (C2, C3 and C16) for the solution of the Euler system with the initial condition (6.47), $I \times J = 100 \times 100$, $T = 0.3$.

Bigger CNN and ability to generalize on unseen configurations

As can be seen from the previous section, the WENO-DS methods trained using the data according to configuration 3 and configuration 16 are able to generalize very well to unseen problems. The WENO-DS method is able to properly localize the shocks and discontinuities, leading to a better numerical solution. Let us now increase the size of the CNN and use the structures shown in Figures 5.3b, increasing the size of the receptive field and the number of channels, and Figure 5.3c, increasing the number of channels and adding another CNN layer. Experimentally, we found out that only increasing the size of the receptive field and the number of channels leads to similar results as described in the previous sections. In addition, increasing the receptive field makes the WENO-DS computationally more expensive. This is because we need to prepare wider inputs for the CNN, which also need to be projected onto the characteristic fields using left eigenvectors, and the matrix multiplications are more expensive here. On the other hand, if we use the CNN structure described in Figure 5.3c, we obtain a trained WENO-DS method that provides a much better numerical solution even for unseen problems with significantly different initial configurations.

Let us now train the method on two data sets. First, we use the dataset corresponding to configuration 3, train the CNN, and denote the final method as WENO-DS (C3c). Second, we train the CNN on the data set corresponding to configuration 16 and denote the final method as WENO-DS (C16c). We test the methods on even more different configurations and compare the results in Tables 6.8 and 6.9. We use boldface to indicate the configuration on which the method was actually trained.

With the number of configurations listed in the tables, we cover a wide range of possible combinations of contact discontinuities, rarefaction and shock waves. For all of them we use the test examples from the literature, see, e.g. [64]. We treat the possibility with four contact discontinuities with configuration 6: J_{21}^-, J_{32}^-, J_{34}^-, J_{41}^-, two contact discontinuities and two rarefaction waves with configuration 8: $\overleftarrow{R}_{21}$, J_{32}^-, J_{34}^-, $\overleftarrow{R}_{41}$, two shock waves and two contact discontinuities using configuration 14: J_{21}^+, $\overleftarrow{S}_{32}$, J_{34}^-, $\overleftarrow{S}_{41}$ and configuration 11: $\overleftarrow{S}_{21}$, J_{32}^+, J_{34}^+, $\overleftarrow{S}_{41}$. Finally, the combination of contact discontinuities, rarefaction, and shock waves using configuration 18: J_{21}^+, $\overleftarrow{S}_{32}$, J_{34}^+, $\overrightarrow{R}_{41}$, and configuration 19: J_{21}^+, $\overleftarrow{S}_{32}$, J_{34}^-, $\overrightarrow{R}_{41}$.

As one can see, we obtain significant error improvements with both methods. Comparing both methods, even better results are obtained when the CNN was trained on a data on a configuration with four shock waves (configuration 3). Compared to the Table 6.4, the improvement for configuration 3 is smaller but still significant. However, the method is able to generalize much better to unknown configurations. For example, for configuration 14, we obtain an average improvement rate of 1.30 for all four variables. In addition, we use WENO-DS (C3c) to illustrate the density contour plots and absolute pointwise errors in Figures 6.26, 6.27, and 6.28. Again, the difference from configuration 3, which was actually used to train the model, is evident.

The WENO-DS (C3c) method achieves large error improvements not only for problems from the same configuration, but also for problems from significantly different configurations. Since we used a larger CNN, the question is what is the actual numerical cost of these improvements? We illustrate the computational costs in Figure 6.29. As can be seen from the shift of the red dots to the right, the method involves larger computational costs. However, it is still more effective or not worse than the original method in most cases. We would like to emphasize that here we are comparing results with significantly different initial problems than those on which the method was actually trained. Machine learning models are generally not expected to give much better results on unseen problems.

	Configuration 3			Configuration 6			Configuration 8			Configuration 11		
	WENO-Z	**WENO-DS**	**ratio**	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	**0.019232**	**0.015033**	**1.28**	0.038616	0.032696	1.18	0.005711	0.004975	1.15	0.007792	0.006316	1.23
u	**0.019588**	**0.016359**	**1.20**	0.019662	0.016144	1.22	0.008488	0.007396	1.15	0.008003	0.006487	1.23
v	**0.019588**	**0.016359**	**1.20**	0.022582	0.018951	1.19	0.008488	0.007396	1.15	0.007692	0.006282	1.22
p	**0.018666**	**0.015214**	**1.23**	0.010525	0.008821	1.19	0.005350	0.004844	1.10	0.005883	0.004813	1.22

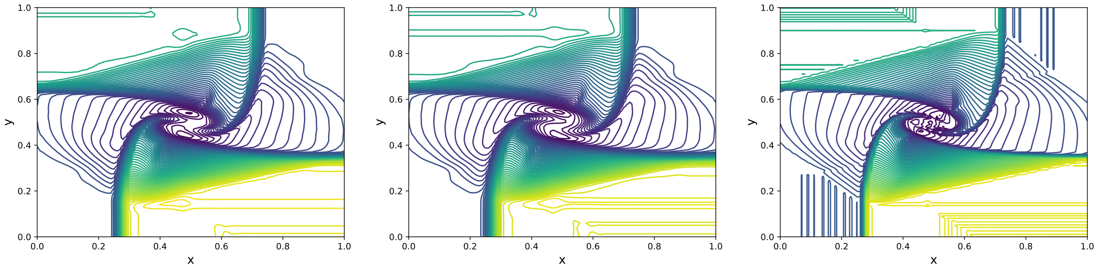
	Configuration 14			Configuration 18			Configuration 19		
	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	0.013169	0.010333	1.27	0.014918	0.012519	1.19	0.014844	0.012390	1.20
u	0.004835	0.003732	1.30	0.003534	0.003063	1.15	0.003749	0.003339	1.12
v	0.021299	0.016512	1.29	0.010315	0.009077	1.14	0.009891	0.008641	1.14
p	0.034996	0.026008	1.35	0.006795	0.005961	1.14	0.006123	0.005393	1.14

Table 6.8: Comparison of L_1 errors of WENO-Z and WENO-DS (C3c) methods for the solution of the Euler system with various initial configurations, $I \times J = 100 \times 100$.

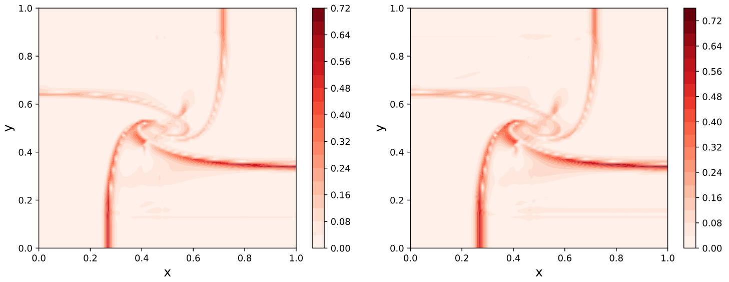
	Configuration 16			Configuration 6			Configuration 8			Configuration 11		
	WENO-Z	**WENO-DS**	**ratio**	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	**0.004834**	**0.004127**	**1.17**	0.038616	0.036329	1.06	0.005711	0.004777	1.20	0.007792	0.007695	1.01
u	**0.005989**	**0.004981**	**1.20**	0.019662	0.019575	1.00	0.008488	0.007056	1.20	0.008003	0.007824	1.02
v	**0.006609**	**0.005776**	**1.14**	0.022582	0.019974	1.13	0.008488	0.007056	1.20	0.007692	0.007482	1.03
p	**0.004236**	**0.003663**	**1.16**	0.010525	0.010216	1.03	0.005350	0.004624	1.16	0.005883	0.006295	0.93

	Configuration 14			Configuration 18			Configuration 19		
	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio	WENO-Z	WENO-DS	ratio
ρ	0.013169	0.011718	1.12	0.014918	0.013447	1.11	0.014844	0.013198	1.12
u	0.004835	0.004042	1.20	0.003534	0.002975	1.19	0.003749	0.003256	1.15
v	0.021299	0.020330	1.05	0.010315	0.009302	1.11	0.009891	0.008796	1.12
p	0.034996	0.036038	0.97	0.006795	0.006535	1.04	0.006123	0.005752	1.06

Table 6.9: Comparison of L_1 errors of WENO-Z and WENO-DS (C16c) methods for the solution of the Euler system with various initial configurations, $I \times J = 100 \times 100$.

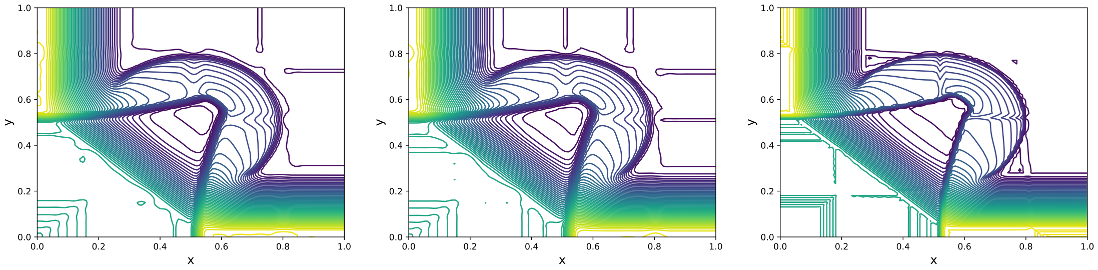


(a) WENO-DS (b) WENO-Z (c) reference solution

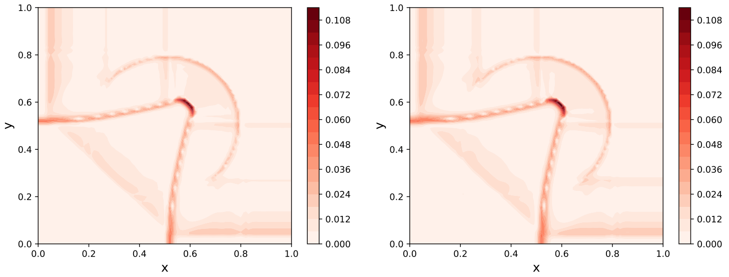


(d) WENO-DS (e) WENO-Z

Figure 6.26: Density contour plots and absolute pointwise errors for the solution of the Riemann problem with initial configuration 6, $I \times J = 100 \times 100$, $T = 0.3$.

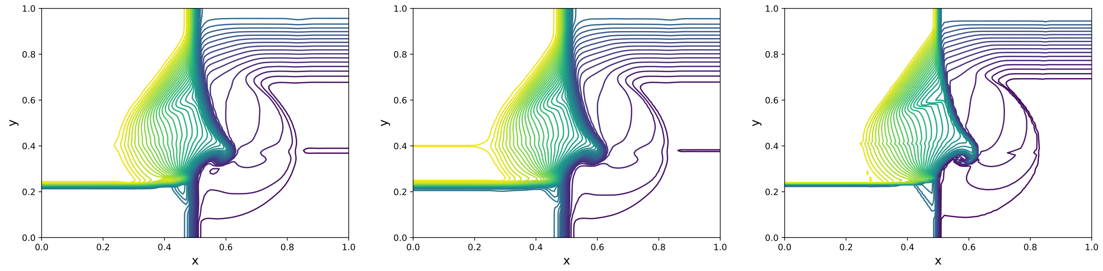


(a) WENO-DS (b) WENO-Z (c) reference solution

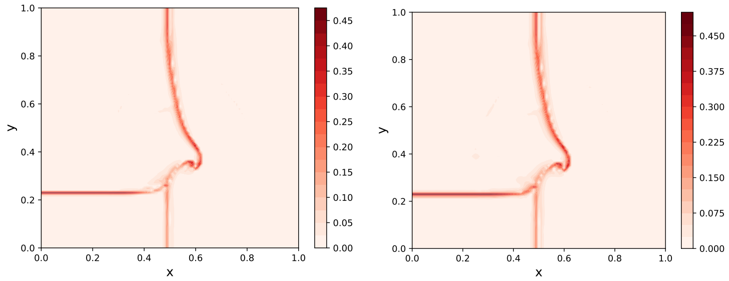


(d) WENO-DS (e) WENO-Z

Figure 6.27: Density contour plots and absolute pointwise errors for the solution of the Riemann problem with initial configuration 8, $I \times J = 100 \times 100$, $T = 0.25$.

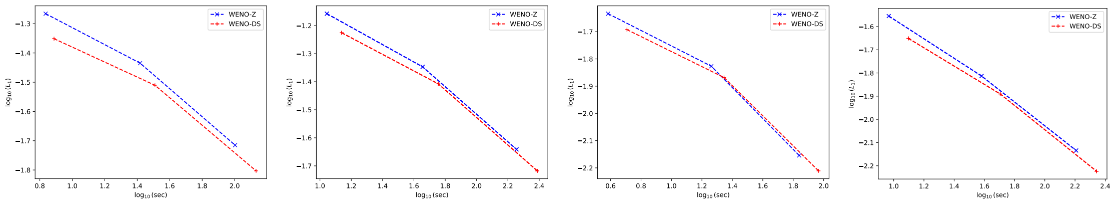


(a) WENO-DS (b) WENO-Z (c) reference solution

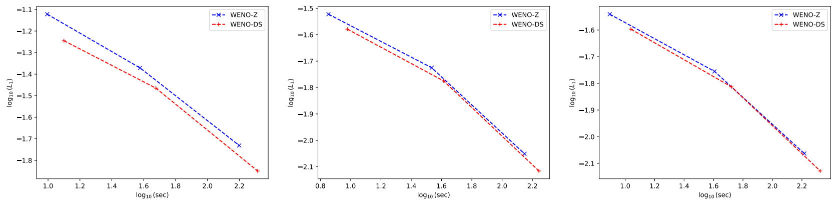


(d) WENO-DS (e) WENO-Z

Figure 6.28: Density contour plots and absolute pointwise errors for the solution of the Riemann problem with initial configuration 19, $I \times J = 100 \times 100$, $T = 0.3$.



(a) Configuration 3 (b) Configuration 6 (c) Configuration 8 (d) Configuration 11



(e) Configuration 14 (f) Configuration 18 (g) Configuration 19

Figure 6.29: Comparison of computational cost against L_1 error of the solution of Riemann problem with various initial configurations using the WENO-Z and WENO-DS (C3c) methods.

7

Chapter 7

Sixth-order WENO scheme for nonlinear degenerate parabolic equations

In this chapter we describe the development of a new modification of WENO method for solving nonlinear degenerate parabolic equations using deep learning techniques. To this end, we modify the smoothness indicators of an existing WENO algorithm that are responsible for measuring the discontinuity of a numerical solution. We do this in such a way that the consistency and convergence of our new WENO-DS method is preserved and can be theoretically proven. Furthermore, we show that the WENO-DS method can be easily applied in more dimensions without the need to retrain the CNN.

We present our results on benchmark examples of nonlinear degenerate parabolic equations, such as the porous medium equation with the Barenblatt solution, the Buckley-Leverett equation and their extensions in two-dimensional space. Here we show that our novel method outperforms the standard WENO method, reliably handles the sharp interfaces and provides good resolution of discontinuities.

We consider nonlinear degenerate parabolic equations of the form

$$\begin{aligned} \frac{\partial}{\partial t} u(\mathbf{x},t) &= \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2} b_i(u(\mathbf{x},t)), \qquad (\mathbf{x},t) \in \Omega \times (0,\infty), \\ u(\mathbf{x},0) &= u_0(\mathbf{x}), \end{aligned} \tag{7.1}$$

where $\mathbf{x} = (x_1, \ldots, x_d)$, with d being the space dimension. The simplest form of (7.1) with $d = 1$ can be represented by

$$\begin{aligned} u_t &= b(u)_{xx}, \\ u(x,0) &= u_0(x), \end{aligned} \tag{7.2}$$

where $b'(u) \geq 0$ and it is possible that $b'(u)$ vanishes for some values of u. In this case, the equation (7.2) degenerates on the u-plane and is not strictly parabolic. We note that such equations occur frequently in applications. For $b(u) = u^m$, the equation (7.2) is written as the *porous medium equation* (PME) [6, 111]:

$$u_t = (u^m)_{xx} = (mu^{m-1}u_x)_x, \qquad m > 1, \tag{7.3}$$

which models the flow of an isentropic gas through a homogeneous porous medium. In this case, the density of the gas u is then a solution of the PME and the pressure of the gas is $p = u^{m-1}$. Since the diffusivity $D(u) = mu^{m-1}$ approaches zero when $u \to 0$, the equation (7.3) degenerates at $u = 0$, leading to the well-known phenomenon of finite speed of propagation and sharp fronts [6, 24, 111].

In general, a classical solution, i.e., a solution which is twice continuously differentiable with respect to x, could not exist even in the case of a smooth initial condition. Therefore, the so-called weak solution must be considered, which is investigated e.g. in [3, 78, 109]. In this chapter, we use the WENO method for solving this type of equation.

7.1 The WENO scheme

We first describe the general WENO discretization to solve (7.2) as developed in [74] and later in [35]. We introduce the uniform grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0, \dots, I$, with the cell boundaries $x_{i+\frac{1}{2}} = x_i + \frac{\Delta x}{2}$. The semi-discrete formulation of (7.2) can be written as

$$\frac{du_i(t)}{dt} = \frac{\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}}{\Delta x^2}, \tag{7.4}$$

where $u_i(t)$ approximates pointwise $u(x_i, t)$ and the numerical flux $\hat{f}_{i+\frac{1}{2}}$ is chosen such that for all sufficiently smooth u

$$\frac{1}{\Delta x^2}\Big(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\Big) = \big(b(u)\big)_{xx}\big|_{x=x_i} + O(\Delta x^6), \tag{7.5}$$

with sixth-order of accuracy. Following [74] if we implicitly define a function h by

$$b\big(u(x)\big) = \frac{1}{\Delta x^2} \int\limits_{x-\frac{\Delta x}{2}}^{x+\frac{\Delta x}{2}} \Bigg(\int\limits_{\eta-\frac{\Delta x}{2}}^{\eta+\frac{\Delta x}{2}} h(\xi)\, d\xi \Bigg)\, d\eta, \tag{7.6}$$

then

$$\big(b(u)\big)_{xx} = \frac{1}{\Delta x^2}\big[h(x+\Delta x) - 2h(x) + h(x-\Delta x)\big], \tag{7.7}$$

and with the function

$$g(x) = h\Big(x + \frac{\Delta x}{2}\Big) - h\Big(x - \frac{\Delta x}{2}\Big), \tag{7.8}$$

it holds that

$$\big(b(u)\big)_{xx}\big|_{x=x_i} = \frac{g(x+\frac{\Delta x}{2}) - g(x-\frac{\Delta x}{2})}{\Delta x^2}. \tag{7.9}$$

7.1.1 Sixth-order WENO scheme

Let us now consider a 6-point stencil corresponding to sixth-order discretization:

$$S(i) = \{x_{i-2}, \dots, x_{i+3}\}. \tag{7.10}$$

This will be divided into three candidate substencils given by

$$S(i)^m = \{x_{i-2+m}, \dots, x_{i+1+m}\}, \qquad m = 0, 1, 2. \tag{7.11}$$

On each of these substencils, the numerical flux $\hat{f}^m_{i\pm\frac{1}{2}}$ needs to be calculated. Let $\hat{f}^m(x)$ be the polynomial approximation of $g(x)$ on each of the substencils (7.11). By an evaluation of these polynomials at $x = x_{i+\frac{1}{2}}$ following formulas from [74] can be obtained:

$$\begin{aligned} \hat{f}^0_{i+\frac{1}{2}} &= \frac{b(u_{i-2}) - 3b(u_{i-1}) - 9b(u_i) + 11b(u_{i+1})}{12}, \\ \hat{f}^1_{i+\frac{1}{2}} &= \frac{b(u_{i-1}) - 15b(u_i) + 15b(u_{i+1}) - b(u_{i+2})}{12}, \\ \hat{f}^2_{i+\frac{1}{2}} &= \frac{-11b(u_i) + 9b(u_{i+1}) + 3b(u_{i+2}) - b(u_{i+3})}{12}, \end{aligned} \tag{7.12}$$

and by shifting each index by -1 we obtain the numerical fluxes $\hat{f}^m_{i-\frac{1}{2}}$. The value of a function b at $u(x_i)$ is indicated by $b(u_i) = b(u(x_i))$. The linear combination of the fluxes (7.12) gives the final approximation on the big stencil (7.10) given by

$$\hat{f}_{i+\frac{1}{2}} = \sum_{m=0}^{2} d_m \hat{f}^m_{i+\frac{1}{2}}, \tag{7.13}$$

where d_m are the linear weights, which values are

$$d_0 = -\frac{2}{15}, \qquad d_1 = \frac{19}{15}, \qquad d_2 = -\frac{2}{15}. \tag{7.14}$$

They are also called ideal weights as they would yield the central sixth-order scheme. As can be seen, the linear weights d_0 and d_2 are negative. Therefore, the final WENO scheme may be unstable and a special technique treating the negative weights has to be used [97]. The weights d_m are then split into positive and negative parts, such that it holds

$$d_m = \sigma^+ \gamma^+_m - \sigma^- \gamma^-_m. \tag{7.15}$$

Following [74] we get the values

$$\begin{aligned} \gamma^+_0 &= \frac{1}{21}, \quad \gamma^+_1 = \frac{19}{21}, \quad \gamma^+_2 = \frac{1}{21}, \\ \gamma^-_0 &= \frac{4}{27}, \quad \gamma^-_1 = \frac{19}{27}, \quad \gamma^-_2 = \frac{4}{27}, \end{aligned} \tag{7.16}$$

and

$$\sigma^+ = \frac{42}{15}, \quad \sigma^- = \frac{27}{15}. \tag{7.17}$$

Finally, the numerical flux for the WENO scheme can be approximated by

$$\hat{f}_{i+\frac{1}{2}} = \sum_{m=0}^{2} \omega_m \hat{f}^m_{i+\frac{1}{2}}, \tag{7.18}$$

with

$$\omega_m = \sigma^+ \alpha^+_m - \sigma^- \alpha^-_m, \tag{7.19}$$

$$\alpha^\pm_m = \frac{\tilde{\alpha}^\pm_m}{\sum_{i=0}^{2} \tilde{\alpha}^\pm_i}, \qquad \tilde{\alpha}^\pm_m = \frac{\gamma^\pm_m}{(\epsilon + \beta_m)^2}, \qquad m = 0, 1, 2. \tag{7.20}$$

β_m is referred to as the smoothness indicator, which plays the crucial role in deciding which substencils should be chosen for the final flux approximation and the parameter ϵ is used to prevent the denominator from becoming zero. Moreover, it should be chosen carefully, as it can reduce the order of accuracy of the underlying scheme, which has been studied in [4] or [42].

7.1.2 Smoothness indicators

In this section we analyze the smoothness indicators β_m as proposed in [50]. They are defined as:

$$\beta_m = \sum_{q=1}^{2} \Delta x^{2q-1} \int_{x_i}^{x_{i+1}} \left(\frac{d^q \hat{f}^m(x)}{dx^q} \right)^2 dx, \tag{7.21}$$

with $\hat{f}^m(x)$ being the polynomial approximation in each of three substencils. There is only one difference from equation (5.19), namely that the integration must be over the interval $[x_i, x_{i+1}]$ to satisfy the symmetry property of the parabolic equation. The right-hand side of (7.21) is the sum of the L^2-norms of the first and second derivatives of the interpolation polynomial $\hat{f}^m(x)$ over the interval (x_i, x_{i+1}). This is the total variation for the approximation and thus a good measure of smoothness [50]. The explicit forms of these indicators corresponding to the flux approximation

$\hat{f}_{i+\frac{1}{2}}$ can be obtained as

$$\begin{aligned}
\beta_0 &= \frac{13}{12}\big(b(u_{i-2}) - 3b(u_{i-1}) + 3b(u_i) - b(u_{i+1})\big)^2 \\
&\quad + \frac{1}{4}\big(b(u_{i-2}) - 5b(u_{i-1}) + 7b(u_i) - 3b(u_{i+1})\big)^2, \\
\beta_1 &= \frac{13}{12}\big(b(u_{i-1}) - 3b(u_i) + 3b(u_{i+1}) - b(u_{i+2})\big)^2 \\
&\quad + \frac{1}{4}\big(b(u_{i-1}) - b(u_i) - b(u_{i+1}) + b(u_{i+2})\big)^2, \\
\beta_2 &= \frac{13}{12}\big(b(u_i) - 3b(u_{i+1}) + 3b(u_{i+2}) - b(u_{i+3})\big)^2 \\
&\quad + \frac{1}{4}\big(- 3b(u_i) + 7b(u_{i+1}) - 5b(u_{i+2}) + b(u_{i+3})\big)^2,
\end{aligned} \tag{7.22}$$

and the Taylor expansion at x_i gives

$$\begin{aligned}
\beta_0 &= b_{xx}^2 \Delta x^4 + b_{xx}^2 f_{xxx} \Delta x^5 + \Big(\frac{4}{3} b_{xxx}^2 - \frac{1}{3} b_{xx} b_{xxxx}\Big)\Delta x^6 \\
&\quad + \Big(\frac{1}{4} b_{xx} b_{xxxxx} - \frac{5}{4} b_{xxx} b_{xxxx}\Big)\Delta x^7 + O\big(\Delta x^8\big), \\
\beta_1 &= b_{xx}^2 \Delta x^4 + b_{xx}^2 b_{xxx} \Delta x^5 + \Big(\frac{4}{3} b_{xxx}^2 + \frac{2}{3} b_{xx} b_{xxxx}\Big)\Delta x^6 \\
&\quad + \Big(\frac{1}{4} b_{xx} b_{xxxxx} + \frac{17}{12} b_{xxx} b_{xxxx}\Big)\Delta x^7 + O\big(\Delta x^8\big), \\
\beta_2 &= b_{xx}^2 \Delta x^4 + b_{xx}^2 b_{xxx} \Delta x^5 + \Big(\frac{4}{3} b_{xxx}^2 - \frac{1}{3} b_{xx} b_{xxxx}\Big)\Delta x^6 \\
&\quad + \Big(-\frac{3}{4} b_{xx} b_{xxxxx} + \frac{37}{12} b_{xxx} b_{xxxx}\Big)\Delta x^7 + O\big(\Delta x^8\big).
\end{aligned} \tag{7.23}$$

Here (and further), the short notation $b_x = b(u_i)_x$ is used.

The smoothness indicators are designed to approach zero for smooth parts of the solution, so that the nonlinear weights ω_m approach the ideal weights d_m. When the substencil S^m contains a discontinuity, the corresponding smoothness indicator β_m is $O(1)$ and the corresponding nonlinear weight ω_m becomes smaller, reducing the contribution of the substencil S^m. For more details and the accuracy analysis we refer the reader to [74]. It has been shown, that for the sixth-order accuracy the following necessary and sufficient conditions have to be satisfied:

$$\begin{aligned}
\omega_m - d_m &= O(\Delta x^3), \\
\omega_0 - \omega_2 &= O(\Delta x^4).
\end{aligned} \tag{7.24}$$

As it was shown in [74], the nonlinear weights (7.19)-(7.20) with smoothness indicators (7.22) do not fulfill the conditions (7.24). Therefore, the mapped function as introduced in [42] was used by Liu et al. [74].

7.1.3 The MWENO scheme

Alternatively, Hajipour and Malek [35] defined new nonlinear weights using

$$\alpha_m^{\pm} = \frac{\tilde{\alpha}_m^{\pm}}{\sum\limits_{i=0}^{2} \tilde{\alpha}_i^{\pm}}, \qquad \tilde{\alpha}_m^{\pm} = \gamma_m^{\pm}\left[1 + \left(\frac{\tau_7}{\beta_m + \epsilon}\right)^2\right], \qquad m = 0, 1, 2, \tag{7.25}$$

and then inserting into (7.19) with

$$\tau_7 = |\beta_0 - \beta_2|. \tag{7.26}$$

From (7.23) it can be seen that

$$\tau_7 = \left|-b_{xx}b_{xxxxx} + \frac{13}{3}b_{xxx}b_{xxxx}\right|\Delta x^7 + O(\Delta x^8). \tag{7.27}$$

It has been shown [35], that using these nonlinear weights the conditions (7.24) are satisfied and the sixth-order accuracy is ensured. Let us note, that this is an analogous formulation to the one used in [13] and described in Section 5.1.3. Here, the WENO-Z method was developed for solving hyperbolic conservation laws.

7.2 Application of deep learning to the sixth-order WENO scheme

Solving nonlinear degenerate parabolic equations is a challenging task in most cases. Not only because of the possible existence of non-smooth solutions or sharp fronts, but also because of the finite propagation speed of the wave fronts. This gives us enough room to improve the existing WENO method. In Chapter 5 new smoothness indicators for the fifth-order WENO-DS scheme were developed using deep learning. They were defined as the product of the original smoothness indicators β_m and perturbations δ_m, where δ_m are the outputs of a particular NN algorithm. We apply this idea and modify the smoothness indicators (7.22) in the same way to improve the sixth-order WENO scheme.

Analogously, we define the new smoothness indicators:

$$\begin{aligned} \beta_{m,i+\frac{1}{2}}^{DS} &= \beta_{m,i+\frac{1}{2}}(\delta_{m,i+\frac{1}{2}} + C), \\ \beta_{m,i-\frac{1}{2}}^{DS} &= \beta_{m,i-\frac{1}{2}}(\delta_{m,i-\frac{1}{2}} + C), \end{aligned} \tag{7.28}$$

with

$$\delta_{0,i+\frac{3}{2}} = \delta_{1,i+\frac{1}{2}} = \delta_{2,i-\frac{1}{2}}, \quad i = 0, \dots, I. \tag{7.29}$$

Here, $\beta_{m,i+\frac{1}{2}}^{DS}$, $m = 0, 1, 2$ are the smoothness indicators corresponding to the numerical fluxes $\hat{f}_{i+\frac{1}{2}}^m$, $m = 0, 1, 2$. For the numerical fluxes $\hat{f}_{i-\frac{1}{2}}^m$, $m = 0, 1, 2$, the smoothness indicators $\beta^{DS}_{m,i-\frac{1}{2}}$, $m = 0, 1, 2$ are used. Defining the smoothness indicators in this way we preserve the conservative property of a scheme, as the values $\hat{f}^m_{i-\frac{1}{2}}$ can be obtained from $\hat{f}^m_{i+\frac{1}{2}}$ by simple index shift.

7.2.1 Accuracy analysis

In this section we analyze the accuracy of the new WENO-DS method for a smooth solutions u. For this purpose, the Assumption 5.2.1 formulated in Chapter 5 will be used. As in this chapter new equations and formulas for nonlinear degenerate parabolic equations were introduced, for the sake of convenience, we formulate an analogous assumption in this chapter as well.

Assumption 7.2.1. *Let us assume, that in each time step there exists a function* $\Phi \in \mathbb{R}^{2k+1} \to \mathbb{R}$*, such that the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *from* (7.28) *in the node* x_i *satisfying* (7.29) *can be expressed as:*

$$\begin{aligned} \delta_{0,i+\frac{1}{2}} &= \Phi(\bar{x}_i - \Delta x) = \Phi(\bar{x}_i) - O(\Delta x), \\ \delta_{1,i+\frac{1}{2}} &= \Phi(\bar{x}_i), \\ \delta_{2,i+\frac{1}{2}} &= \Phi(\bar{x}_i + \Delta x) = \Phi(\bar{x}_i) + O(\Delta x), \end{aligned} \tag{7.30}$$

where

$$\bar{x}_i = (x_{i-k}, x_{i-k+1}, \dots, x_{i+k}). \tag{7.31}$$

Moreover, let us assume that there exists a constant C *independent from time step and discretization, such that it holds* $\Phi(\bar{x}_i) + C > \kappa > 0$ *with* κ *fixed,* $i = 0, \dots, I$.

Theorem 3. *Let the numerical flux of the WENO-DS scheme be given by* (7.12) *and* (7.18) *with the corresponding nonlinear weights given by* (7.19) *and*

$$\alpha^{\pm}_m = \frac{\tilde{\alpha}^{\pm}_m}{\sum\limits_{i=0}^{2} \tilde{\alpha}^{\pm}_i}, \qquad \tilde{\alpha}^{\pm}_m = \gamma^{\pm}_m \left[1 + \left(\frac{\tau_7}{\beta^{DS}_{m,i\pm\frac{1}{2}} + \epsilon}\right)^2\right], \qquad m = 0, 1, 2, \tag{7.32}$$

with $\gamma^{\pm}_m$ *given by* (7.16)*,* $\beta^{DS}_{m,i\pm\frac{1}{2}}$ *defined in* (7.28) *and* τ_7 *defined by* (7.26)*. Let the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *in* (7.28) *satisfy the Assumption 7.2.1. Then, the resulting WENO-DS method* (7.4) *for smooth solutions of the nonlinear degenerate parabolic equation* (7.1) *exhibits a sixth-order accuracy, assuming* $|b_{xx}| \geq \eta > 0$.

Proof. From (7.23) we see that

$$\beta_{m,i\pm\frac{1}{2}} = b^2_{xx} \Delta x^4 + O(\Delta x^5), \tag{7.33}$$

and from (7.27)

$$\tau_7 = O(\Delta x^7). \tag{7.34}$$

Then using Assumption 7.2.1 it holds

$$\begin{aligned} \beta^{DS}_{m,i\pm\frac{1}{2}} &= \beta_{m,i\pm\frac{1}{2}}(\delta_{m,i\pm\frac{1}{2}} + C) = \big(b_{xx}^2\Delta x^4 + O(\Delta x^5)\big)\big(\Phi(\bar{x}_i) + O(\Delta x) + C\big) \\ &= b_{xx}^2 P(\bar{x}_i)\Delta x^4 + O(\Delta x^5), \end{aligned} \tag{7.35}$$

with $P(\bar{x}_i) = \Phi(\bar{x}_i) + C$ and C satisfying the Assumption 7.2.1. Then $P(\bar{x}_i) = O(1)$ is ensured. Assuming $|b_{xx}| \geq \eta > 0$, it holds

$$\frac{\tau_7}{\beta^{DS}_{m,i\pm\frac{1}{2}}} = \hat{D}\Delta x^3 + O(\Delta x^4), \qquad \hat{D} = \frac{|-b_{xx}b_{xxxxxx} + \frac{13}{3}b_{xxx}b_{xxxx}|}{b_{xx}^2 P(\bar{x}_i)}. \tag{7.36}$$

We take $\epsilon = 0$, substitute now this into (7.32) (for simplicity we drop the index $i \pm \frac{1}{2}$) and obtain

$$\tilde{\alpha}^{\pm}_m = \gamma^{\pm}_m\left[1 + \Big(\frac{\tau_7}{\beta^{DS}_m + \epsilon}\Big)^2\right] = \gamma^{\pm}_m\big(1 + O(\Delta x^6)\big), \tag{7.37}$$

and

$$\alpha^{\pm}_m = \frac{\gamma^{\pm}_m\left[1 + \Big(\frac{\tau_7}{\beta^{DS}_m + \epsilon}\Big)^2\right]}{\sum\limits_{i=0}^{2} \gamma^{\pm}_m\big(1 + O(\Delta x^6)\big)}, \tag{7.38}$$

which implies

$$\gamma^{\pm}_m = \alpha^{\pm}_m \frac{1}{1 + \Big(\frac{\tau_7}{\beta^{DS}_m + \epsilon}\Big)^2} \sum_{i=0}^{2} \gamma^{\pm}_m\big(1 + O(\Delta x^6)\big) = \alpha^{\pm}_m + O(\Delta x^6), \tag{7.39}$$

where we used $\sum_{i=0}^{2} \gamma^{\pm}_m = 1$.

We investigate now the conditions (7.24). Using (7.39), inserting into (7.15) and using (7.19) we fulfill the first condition:

$$d_m = \sigma^+\big(\alpha^+_m + O(\Delta x^6)\big) - \sigma^-\big(\alpha^-_m + O(\Delta x^6)\big) = \omega_m + O(\Delta x^6). \tag{7.40}$$

Finally, realizing that $\gamma^{\pm}_0 = \gamma^{\pm}_2$, the second condition is also fulfilled:

$$\begin{aligned} \omega_0 - \omega_2 &= \sigma^+\alpha^+_0 - \sigma^-\alpha^-_0 - \sigma^+\alpha^+_2 + \sigma^-\alpha^-_2 = \sigma^+\big(\gamma^+_0 + O(\Delta x^6)\big) \\ &- \sigma^-\big(\gamma^-_0 + O(\Delta x^6)\big) - \sigma^+\big(\gamma^+_2 + O(\Delta x^6)\big) + \sigma^-\big(\gamma^-_2 + O(\Delta x^6)\big) \\ &= O(\Delta x^6) \end{aligned} \tag{7.41}$$

and the sixth-order accuracy of the WENO-DS method for smooth solutions of nonlinear degenerate parabolic equation (7.1) is ensured. □

Theorem 4. *Let us compute the multipliers* $\delta_{m,i\pm\frac{1}{2}}$ *from* (7.28) *in the node* x_i

satisfying (7.29) *using a one-dimensional convolutional neural network with vector* $\bar{b}(\bar{u}_i)$ *defined by*

$$\begin{aligned} \bar{b}(\bar{u}_i) &= (b(u(x_{i-k})), b(u(x_{i-k+1})), \ldots, b(u(x_{i+k}))), \\ \bar{u}_i &= \bar{u}(\bar{x}_i) = (u(x_{i-k}), u(x_{i-k+1}), \ldots, u(x_{i+k})) \end{aligned} \tag{7.42}$$

as input. Further, let all hidden layers of this neural network be differentiable functions and let the activation function in the last layer of the CNN be bounded from below. Then for these multipliers $\delta_{m,i\pm\frac{1}{2}}$ *the Assumption 7.2.1 holds.*

Proof. Let $2k+1$ be the size of the receptive field of the CNN. Let $F(\cdot) \in \mathbb{R}^{2k+1} \to \mathbb{R}$ be the convolutional neural network function:

$$F\big(\bar{b}(\bar{u}_i)\big) = \mathrm{CNN}\big(\bar{b}(\bar{u}_i)\big). \tag{7.43}$$

Then, we can define $\Phi = F \circ \bar{b} \circ \bar{u}$ and it holds

$$\begin{aligned} \delta_{0,i+\frac{1}{2}} &= (F \circ \bar{b} \circ \bar{u})(\bar{x}_i - \Delta x) = \Phi(\bar{x}_i - \Delta x), \\ \delta_{1,i+\frac{1}{2}} &= (F \circ \bar{b} \circ \bar{u})(\bar{x}_i) = \Phi(\bar{x}_i), \\ \delta_{2,i+\frac{1}{2}} &= (F \circ \bar{b} \circ \bar{u})(\bar{x}_i + \Delta x) = \Phi(\bar{x}_i + \Delta x). \end{aligned} \tag{7.44}$$

Moreover, as $F, \bar{b}, \bar{u}$ are all differentiable functions, also Φ is a differentiable function and (7.30) holds.

As the last activation function is bounded from below, there exists ξ such that $\Phi(\bar{x}_i) > \xi$, and any $C > \kappa + \max(-\xi, 0)$ satisfies the second part of Assumption 7.2.1 for arbitrary $\kappa > 0$. □

7.3 Structure of neural network

Analogously to the case of hyperbolic conservation laws, we use CNN for the enhanced WENO-DS scheme. Detailed discussion about the advantages of this NN has already been done in Section 5.3. Also in this case, we experimentally found out that the usage of ELU and Softplus activation functions gives the best results. We set $C = 0.1$ in (7.28) and the value of ϵ in (7.32) to 10^{-13}. For detailed discussions about the constant C we refer to Section 5.3.

7.4 The training procedure

In this chapter we use the training procedure as described in Section 6.1.1. In our experiments we use the probability $\varphi = 0.1$ and set the maximum number of opened problems to 200.

To improve the gradient propagation to the lower layers, we use the residual learning framework [41] as mentioned in Section 1.2.6. It may happen that when using a deeper NN, its effectiveness is compromised, but this effect is not due to overfitting, as reported in [41].

To update the weights of the CNN we use again the Adam optimizer [55]. The optimizer parameters will be specified for each of the equation classes separately. As the default loss function we use the mean square error

$$\mathrm{LOSS}_{\mathrm{MSE}}(u) = \frac{1}{I} \sum_{i=0}^{I} (\hat{u}_i^n - u_i^{n,\mathrm{ref}})^2, \tag{7.45}$$

where $\hat{u}_i^n$ is a numerical approximation of $u(x_i, t_n)$ obtained by WENO-DS and $u_i^{n,\mathrm{ref}}$ denotes the corresponding reference solution. For the implementation we use again Python with the library PyTorch [79].

7.5 Two-dimensional implementation

Here we consider the two-dimensional form of (7.1):

$$u_t = b_1(u)_{xx} + b_2(u)_{yy}. \tag{7.46}$$

The procedure described in Sections 7.1 and 7.2 can be easily applied dimension-by-dimension to obtain the approximations of numerical fluxes $\hat{f}_{i+\frac{1}{2},j}$ and $\hat{k}_{i,j+\frac{1}{2}}$, such that it holds

$$\begin{aligned} \frac{1}{\Delta x^2}\big(\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}\big) &= \big(b_1(u)\big)_{xx}\big|_{(x_i,y_j)} + O\big(\Delta x^6\big), \\ \frac{1}{\Delta y^2}\big(\hat{k}_{ij+\frac{1}{2}} - \hat{k}_{i,j-\frac{1}{2}}\big) &= \big(b_2(u)\big)_{yy}\big|_{(x_i,y_j)} + O\big(\Delta y^6\big), \end{aligned} \tag{7.47}$$

using the uniform grid with nodes (x_i, y_j), $\Delta x = x_{i+1} - x_i$, $\Delta y = y_{j+1} - y_j$, $i = 0, \dots, I$, $j = 0, \dots, J$. The corresponding semi-discrete form of (7.46) takes the form

$$\frac{du_{i,j}(t)}{dt} = \frac{\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}}{\Delta x^2} + \frac{\hat{k}_{i,j+\frac{1}{2}} - \hat{k}_{i,j-\frac{1}{2}}}{\Delta y^2}, \tag{7.48}$$

where $u_{i,j}(t)$ is the numerical approximation to the point value $u(x_i, y_j, t)$.

In our numerical examples we train the CNN on one-dimensional data, as well as on two-dimensional data. We compare the results and we are interested if the method trained on two-dimensional data would bring significantly better numerical approximations, or if the training on one-dimensional data would be sufficient.

7.6 Numerical results

In this section, we present the numerical results to show the efficiency of the proposed numerical scheme WENO-DS based on the CNN algorithm. We use the nonlinear weights (7.25), replacing β_m with β_m^{DS} (7.28). This is done to discretize the diffusion term and for the discretization of the advection term, which later appears in the examples, we use an analogous procedure as described in Chapter 5.

For the system of ODEs resulting from (7.4) we use a third-order TVD Runge-Kutta method [100] given by

$$\begin{aligned} u^{(1)} &= u^n + \Delta t L(u^n), \\ u^{(2)} &= \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}), \\ u^{n+1} &= \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}), \end{aligned} \tag{7.49}$$

where u^n is the numerical solution at the time step n and $L = \frac{1}{\Delta x^2}(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}})$.

As it was already mentioned in Chapter 5, the multipliers update could be done only once per time step, namely in the first Runge-Kutta stage, supposing that the smoothness of the solution does not change significantly within one time step. We performed a large number of numerical experiments and this approach exhibited to be much more effective, as the costly evaluation of CNN needs to be done only once per time step instead of three times per time step. This has only small impact on error, but we can save more computational time, which leads to better numerical performance. So in all our numerical examples we proceed as follows. During the training, we update the multipliers in each Runge-Kutta stage, taking use of all computed values. After the training, when the final WENO-DS scheme is obtained and the concrete problem needs to be computed, we use the same vector of multipliers in the second and third Runge-Kutta stage.

For the solving procedure, we use the time step of the one-dimensional problems

$$u_t + f(u)_x = b(u)_{xx}, \tag{7.50}$$

such that

$$\frac{0.4}{\Delta t} = \frac{\max_u |f'(u)|}{\Delta x} + \frac{\max_u |b'(u)|}{\Delta x^2}. \tag{7.51}$$

For two-dimensional problems

$$u_t + f_1(u)_x + f_2(u)_y = b_1(u)_{xx} + b_2(u)_{yy}, \tag{7.52}$$

the time step is set as

$$\frac{0.4}{\Delta t} = \frac{\max_u |f_1'(u)|}{\Delta x} + \frac{\max_u |f_2'(u)|}{\Delta y} + \frac{\max_u |b_1'(u)|}{\Delta x^2} + \frac{\max_u |b_2'(u)|}{\Delta y^2}. \tag{7.53}$$

In the following examples we provide a detailed comparison of WENO-DS with standard schemes. We compare the WENO-DS method with the MWENO method in the examples where only a parabolic term occurs in the PDEs. Where a hyperbolic term also occurs, we use the MWENO method as the original method in combination with the WENO-Z method [13], which is used to approximate the hyperbolic term.

In the presented tables we compare the L_∞ and L_2 errors. As 'ratio' we denote the error of the MWENO method, resp. MWENO method combined with WENO-Z, divided by the error of WENO-DS (rounded to 2 decimal points).

7.6.1 The porous medium equation with the Barenblatt solution

As the first example we apply the CNN algorithm to enhance the numerical solution of the PME (7.2) with (7.3).

The *Barenblatt solution* [8, 116] is a weak solution of the PME with the explicit form

$$B_m(\mathbf{x},t) = t^{-\alpha}\left[\left(1 - k|\mathbf{x}|^2 t^{-\frac{2\alpha}{d}}\right)^+\right]^{\frac{1}{m-1}}, \quad t > 0, \quad \mathbf{x} \in \Omega \subseteq \mathbb{R}^d, \quad m > 1, \tag{7.54}$$

where $v^+ = \max(v,0)$ and $k = \frac{\alpha(m-1)}{2md}$ with $\alpha = \frac{d}{(m-1)d+2}$. For $d = 1$, the compact support of this Barenblatt solution is the interval $[-a_m(t), a_m(t)]$, where

$$a_m(t) = \sqrt{\frac{2m}{\alpha(m-1)}}\, t^\alpha, \tag{7.55}$$

with $\alpha = \frac{1}{m+1}$. The solution is not differentiable at the interface points $x = \pm a_m(t)$ [75].

In our numerical experiments, we take as initial condition the Barenblatt solution (7.54) at time $t = 1$. We use zero Dirichlet boundary conditions $u(\pm 6, t) = 0$ for $t > 1$ and divide the computational domain into 64 uniform cells.

For the training, we proceed as described in Section 7.4. When a new problem should be selected from a data set, an exponent m in PME (7.3) is chosen randomly such that $m \in \mathcal{U}(2,8)$. In this way, we cover a wide range of different problems and the final numerical scheme can be reliably used for different values of m. For training, we fix $T = 1.4$. We use a rather small CNN with only three hidden layers. The structure is described in Figure 7.1, where also the number of channels and the kernel size can be found. Let us note, that the arrow connecting two (+) blocks represents the residual learning framework mentioned in Section 7.4.

Similarly to Example 5.6.2 we use the adapted loss function to match the training

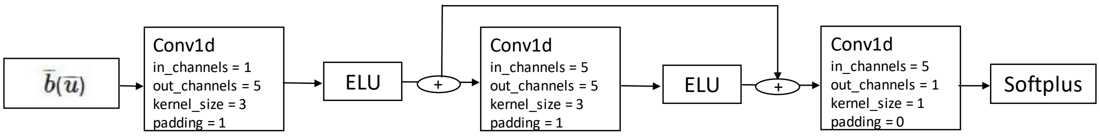


Figure 7.1: A structure of the CNN used for the porous medium equation.

contribution from very small loss problems to large loss problems:

$$\mathrm{LOSS}_{\mathrm{AD}}(u) = \frac{1}{10} 10^{|\lceil \log_{10}(\mathrm{LOSS}_{\mathrm{MSE}}(u)) \rceil|}\, \mathrm{LOSS}_{\mathrm{MSE}}(u), \tag{7.56}$$

where $\mathrm{LOSS}_{\mathrm{MSE}}(u)$ is defined in (7.45) and a reference solution is computed from (7.54). This makes all loss values to be of the order 10^{-2}. To update the weights we use the Adam optimizer with learning rate 0.001.

For testing on a validation set, we use just simple $\mathrm{LOSS}_{\mathrm{MSE}}(u)$ defined by (7.45) as corresponding validation metric. We present the history of the rescaled values

$$\mathrm{LOSS}^{*}_{\mathrm{MSE}}(u) = \frac{\mathrm{LOSS}^{l}_{\mathrm{MSE}}(u)}{\max\limits_{l=0,\dots,L}\left(\mathrm{LOSS}^{l}_{\mathrm{MSE}}(u)\right)}, \qquad l = 0, \dots, L, \tag{7.57}$$

for the problems from the validation set in Figure 7.2, where L denotes the total number of training steps. Here we rescaled the values for each validation problem to be in the interval $[0, 1]$. We tested our model every 100 training steps and the validation metric was evaluated at time $T = 2$. The validation set contains PME problems with different exponents m generated randomly.

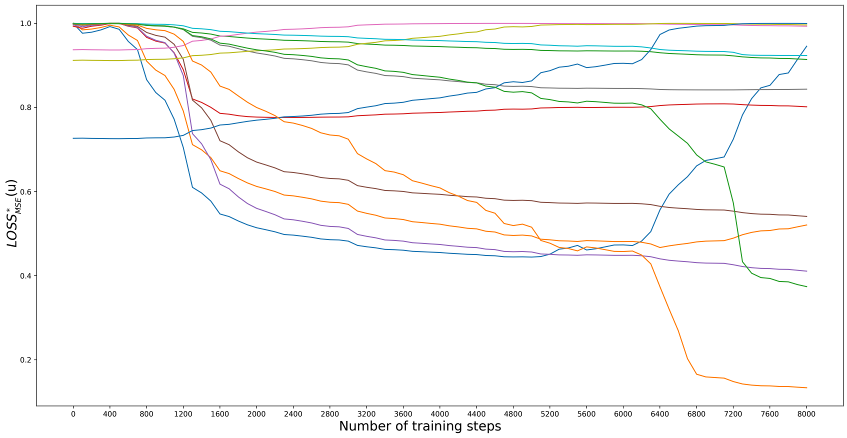
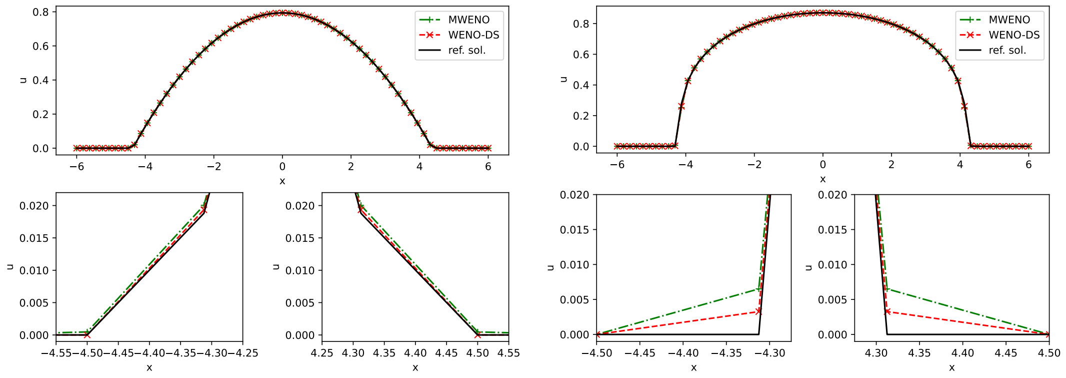


Figure 7.2: Training evolution corresponding to PME: The values (7.57) for different validation problems evaluated each 100 training steps.

In some cases, we see that the values increase slightly as the number of iterations increases. This is because we want to optimize the method for a wide range of parameters m and also over the entire time domain. However, we conclude that in most cases, which we demonstrate later in the tables and figures, the improvement outweighs a slight increase in the error that occurs in a rather small number of cases.

We take the model obtained after the 6800th training step as our final WENO-DS scheme. We see, that longer training leads to the further increase of the validation metric values for some problems, so the suboptimal results would be obtained.

We show the results on problems from the test set. These were not in the training or validation set. In Figure 7.3 we present the solution of the PME for $m = 2, 4$. We observe that WENO-DS yields a better solution in the regions where discontinuity occurs. This also affects the L_∞ and L_2 errors, whose values we compare in Table 7.1. We compare the errors for different parameters m and T and highlight the best performing WENO method in bold. We realize that our new method outperforms the MWENO method in most cases.



(a) $m = 2$

(b) $m = 4$

Figure 7.3: Comparison of the MWENO and WENO-DS methods for the numerical solution of the PME with various parameter values m, $I = 64$.

Finally, we demonstrate the theoretically proven sixth-order accuracy for a heat equation with smooth initial condition given by

$$u_t = u_{xx}, \quad u(x,0) = \sin(x), \quad -\pi \leq x \leq \pi, \quad 0 \leq t \leq 1. \tag{7.58}$$

The exact solution for this example is

$$u(x,t) = e^{-t} \sin x, \tag{7.59}$$

and we take the Dirichlet boundary conditions from the exact solution for this case. The results can be found in Table 7.2 and we observe that with increasing number of space steps, the sixth-order accuracy is ensured. Let us note, that we used for these results the same WENO-DS scheme which was an output of the learning procedure for the PME with the Barenblatt solution. We also did not retrain the CNN for different values of I.

	L_∞			L_2		
m	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
2	0.003257	**0.001185**	2.75	0.002689	**0.001012**	2.66
3	0.017395	**0.014086**	1.23	0.011163	**0.008894**	1.26
4	0.045135	**0.040594**	1.11	0.028080	**0.025064**	1.12
5	0.112800	**0.103776**	1.09	0.069098	**0.063693**	1.08
6	0.177022	**0.169391**	1.05	0.108670	**0.104124**	1.04
7	**0.088695**	0.089579	0.99	**0.057645**	0.058244	0.99
8	**0.175060**	0.179941	0.97	**0.109320**	0.111882	0.98

(a) $T = 1.2$

	L_∞			L_2		
m	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
2	0.004877	**0.003597**	1.36	0.003013	**0.002379**	1.27
3	0.010907	**0.007652**	1.43	0.008057	**0.005233**	1.54
4	0.032591	**0.029520**	1.10	0.020487	**0.018448**	1.11
5	0.104031	**0.097498**	1.07	0.063717	**0.059737**	1.07
6	0.219481	**0.212963**	1.03	0.134668	**0.130745**	1.03
7	0.028863	**0.015848**	1.82	0.018625	**0.010251**	1.82
8	**0.013782**	0.013900	0.99	**0.010280**	0.011442	0.90

(b) $T = 1.5$

	L_∞			L_2		
m	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
2	0.001235	**0.001179**	1.05	0.000952	**0.000933**	1.02
3	0.058471	**0.056152**	1.04	0.036008	**0.034656**	1.04
4	0.026741	**0.017827**	1.50	0.016951	**0.011218**	1.51
5	0.101398	**0.092330**	1.10	0.062115	**0.056666**	1.10
6	0.201053	**0.193171**	1.04	0.123476	**0.118773**	1.04
7	0.052631	**0.040715**	1.29	0.033208	**0.025414**	1.34
8	0.043306	**0.037332**	1.16	0.027796	**0.023919**	1.16

(c) $T = 2$

Table 7.1: Comparison of L_∞ and L_2 errors of MWENO and WENO-DS methods for the solution of the PME with various parameter m and T, $I = 64$.

	MWENO		WENO-DS		MWENO		WENO-DS	
I	L_∞	Order	L_∞	Order	L_2	Order	L_2	Order
10	2.627199e-05	-	7.095802e-06	-	3.844220e-05	-	1.036139e-05	-
20	3.054176e-07	6.426599	2.211357e-07	5.003962	4.936423e-07	6.283081	3.548918e-07	4.867694
40	4.419994e-09	6.110595	4.325358e-09	5.675969	7.496621e-09	6.041082	7.310297e-09	5.601305
80	7.274142e-11	5.925123	7.267770e-11	5.895163	1.260358e-10	5.894334	1.258981e-10	5.859601
160	1.198763e-12	5.923158	1.198652e-12	5.922028	2.100673e-12	5.906838	2.100448e-12	5.905416

Table 7.2: L_∞ and L_2 errors with convergence order of MWENO and WENO-DS on equation (7.58).

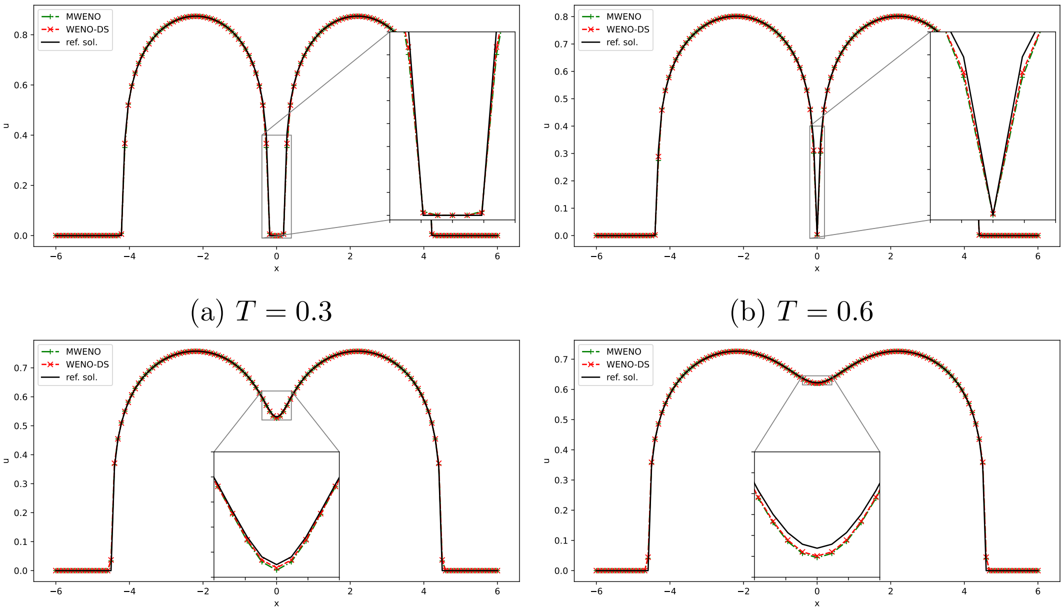
7.6.2 Interaction of two boxes for the porous medium equation

In this example, we consider the collision of two boxes for the PME (7.3). This is a test example presented e.g. in [51, 74]. Considering u as temperature, this model describes how the temperature changes when two hot spots are placed in the computational domain.

We apply the WENO-DS method from Section 7.6.1 in this example. To show its generalizability we did not retrain the CNN for this type of initial data and apply the method directly. First, we take the initial condition for the two-box solution with the same height

$$u(x,0) = \begin{cases} 1, & x \in (-3.7,-0.7) \cup (0.7,3.7), \\ 0, & \text{otherwise}, \end{cases} \tag{7.60}$$

and $m = 5$ in (7.3). We use the computational domain $[-6,6]$ and zero Dirichlet boundary conditions $u(\pm 6,t) = 0$ for $t > 0$. The computational domain is divided into $I = 128$ uniform cells. We present the solution for multiple instances of T in Figure 7.4. We present the solution of the WENO-DS method and the MWENO method in comparison with the reference solution obtained by the MWENO method with $I = 1024$.



(a) $T = 0.3$ (b) $T = 0.6$

(c) $T = 0.9$ (d) $T = 1.2$

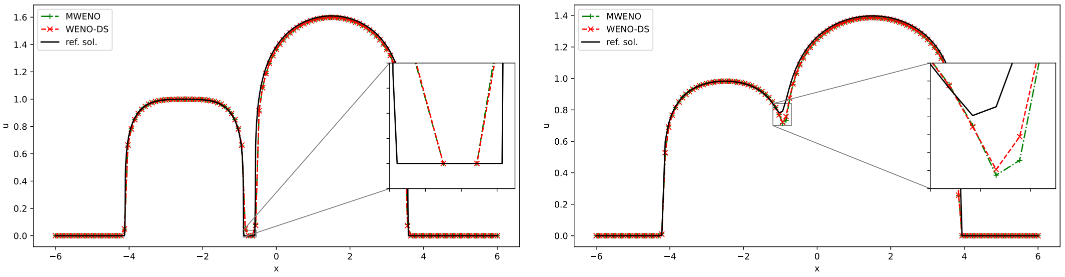
Figure 7.4: Interaction of two boxes for the PME (same height), $I = 128$.

Next, we consider the initial condition for the two-box solution with different height

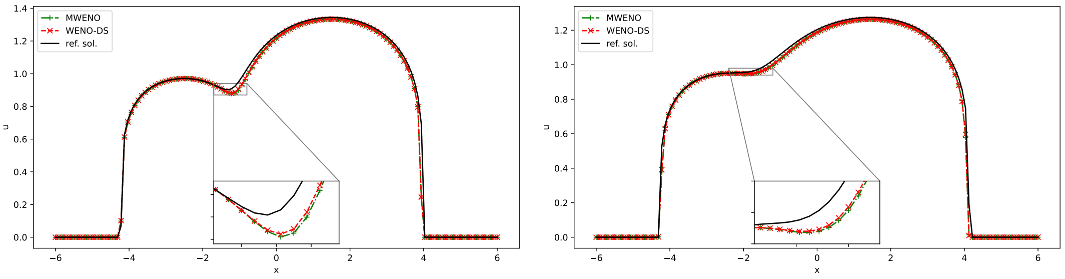
$$u(x,0) = \begin{cases} 1, & x \in (-4,-1), \\ 2, & x \in (0,3), \\ 0, & \text{otherwise}, \end{cases} \tag{7.61}$$

and $m = 6$ in (7.3). We impose zero Dirichlet boundary conditions $u(\pm 6, t) = 0$ for $t > 0$ and divide the computational domain into $I = 128$ uniform cells. We again compare both methods with the reference solution for several instances of T in Figure 7.5.

We find that the WENO-DS method is able to capture the sharp interface very well and in most cases easily outperforms the MWENO method near discontinuities even without additional retraining of the CNN.



(a) $T = 0.02$ (b) $T = 0.06$



(c) $T = 0.08$ (d) $T = 0.12$

Figure 7.5: Interaction of two boxes for the PME (different height), $I = 128$.

7.6.3 The convection-diffusion Buckley-Leverett equation

In the next example we solve the convection-diffusion Buckley–Leverett equation of the form

$$u_t + f(u)_x = \epsilon \left(\nu(u) u_x\right)_x, \qquad \epsilon\, \nu(u) \geq 0. \tag{7.62}$$

This is a prototype model for oil reservoir simulations (two-phase flow). In our test we choose $\epsilon = 0.01$ and the flux function

$$f(u) = \frac{u^2}{u^2 + a(1-u)^2} \left(1 - g(1-u)^2\right), \tag{7.63}$$

where $a < 1$ is a constant representing the ratio of the viscosities of the two fluids and g is a gravitational effect. Usually, $\nu(u)$ vanishes at some points so the equation (7.62) is a degenerate parabolic equation. Moreover, the sign of $f'(u)$ changes its sign, so the handling of the flux is more complicated. We choose

$$\nu(u) = \begin{cases} 4u(1-u), & 0 \leq u \leq 1, \\ 0, & \text{otherwise,} \end{cases} \tag{7.64}$$

so we obtain the parabolic term in a form

$$b(u) = \begin{cases} 0, & u < 0, \\ \epsilon\,(2u^2 - \frac{4}{3}u^3), & 0 \leq u \leq 1, \\ \frac{2}{3}\epsilon, & u > 1. \end{cases} \tag{7.65}$$

The initial condition reads

$$u(x,0) = \begin{cases} 0, & 0 \leq x \leq 1 - \frac{1}{\sqrt{2}}, \\ 1, & 1 - \frac{1}{\sqrt{2}} < x \leq 1, \end{cases} \tag{7.66}$$

and we divide the computational domain into 128 uniform cells.

In the training we proceed as described in Section 7.4. As there exists no analytical solution in this case, we first create our data set, where we compute the reference solutions on a fine grid for the equation (7.62). In this data set we consider the constants $a \in \mathcal{U}[0.1, 0.95]$ and $g \in \mathcal{U}[0, 6]$, divide the computational domain $[0, 1]$ into 1024 uniform cells and compute the solution up to time $T = 0.1$. We use the MWENO method (Section 7.1.3) combined with the WENO-Z method (Section 5.1.3) for the computation of these reference solutions.

We use in this example the same CNN structure as in the previous example, which can be found in Figure 7.1. In the training we optimize not only the WENO-DS method to approximate the parabolic term, but also the WENO-DS to approximate the hyperbolic term. The structure of the CNN remains the same for both cases. We run the training for the total number of 12000 training steps, use the loss function (7.45) and the Adam optimizer with the learning rate 10^{-4}.

We created a validation set with 12 different combinations of a and g generated randomly. On this set, we tested our model every 200 training steps. As a validation metric, we used the same loss function (7.45). The Figure 7.6 shows how the value of the loss function changes as the number of training steps increases. We scaled the loss values again according to equation (7.57) so that they are in $[0, 1]$. We see, there is one problem, for which the best error value after around 6000th training step is obtained. However, for most of the validation problems is the optimal value obtained after around 10000th training step. We choose the model obtained after 10000th training step and present the results computed with this model.

We present the numerical solutions of the Buckley-Leverett equation in Figure 7.7.

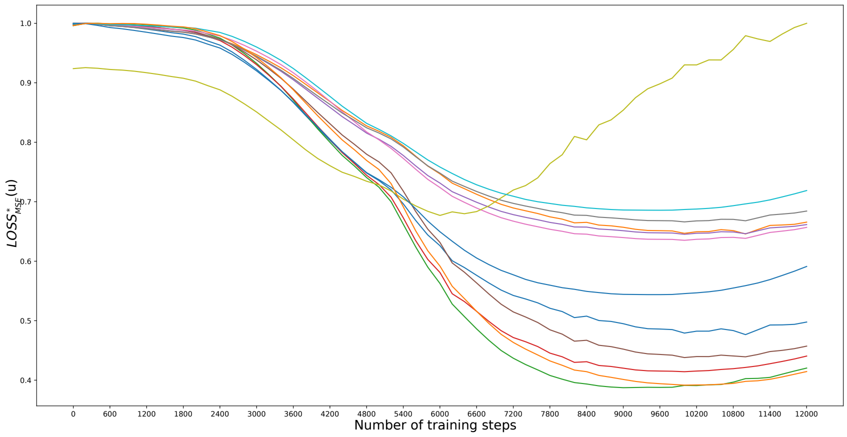
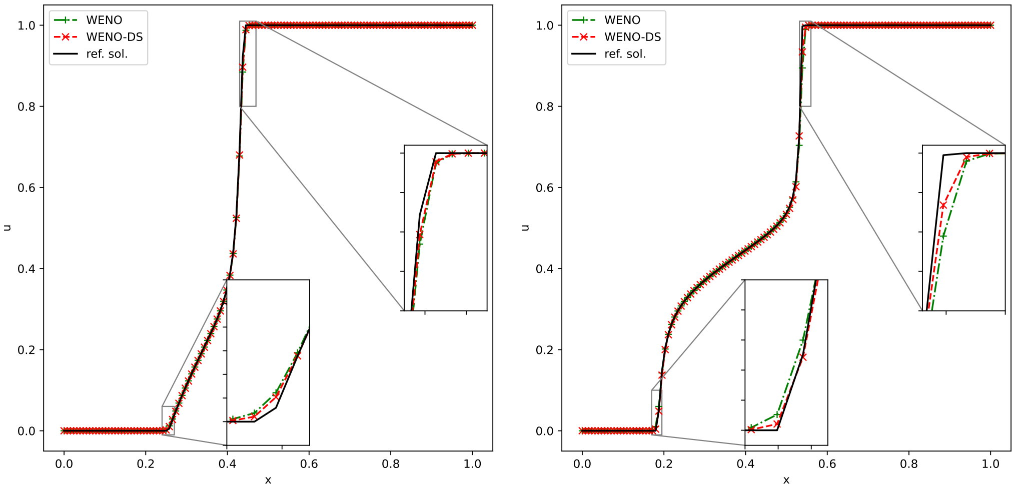


Figure 7.6: Training evolution corresponding to Buckley-Leverett equation: The values (7.57) for different validation problems evaluated each 200 training steps.

We observe that our scheme provides a high quality of numerical solutions for both of these problems. Further, we compare the L_∞ and L_2 errors of the problems from the test set with various parameters a and g in Table 7.3. We see, that in all cases our method provides significantly smaller errors.



(a) $a = 1$ and $g = 0$ (b) $a = 1$ and $g = 5$

Figure 7.7: Comparison of the original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the numerical solution of the Buckley-Leverett equation with various parameters a and g, $T = 0.1$, $I = 128$.

Further, in Table 7.4 we compare the method for different values of final time T. We see, that in all presented examples, WENO-DS outperforms the standard WENO scheme. Finally, we also present the results using WENO-DS for initial value problems with parameters outside of training intervals. During training and validation, we used $g \in \mathcal{U}[0, 6]$ and in Table 7.5 also different values of g can be found. As it can be seen, WENO-DS exhibits in almost all cases smaller errors, which justifies its ability to generalize also for parameters outside of the training

		L_∞			L_2		
a	g	WENO	WENO-DS	ratio	WENO	WENO-DS	ratio
1	5	0.102771	**0.063320**	1.62	0.009442	**0.006067**	1.56
1	0	0.037256	**0.025136**	1.48	0.003709	**0.002678**	1.39
1	3	0.081126	**0.054823**	1.48	0.007497	**0.005241**	1.43
0.75	5	0.065215	**0.015867**	4.11	0.006346	**0.002875**	2.21
0.75	4	0.077982	**0.045051**	1.73	0.007096	**0.004780**	1.48
0.5	5	0.086089	**0.042398**	2.03	0.008228	**0.005991**	1.37
0.5	2	0.045176	**0.028982**	1.56	0.004495	**0.003139**	1.43
0.5	1	0.030054	**0.020181**	1.49	0.003729	**0.002869**	1.30
0.3	3	0.035122	**0.022353**	1.57	0.004715	**0.003156**	1.49
0.25	4	0.083372	**0.042914**	1.94	0.007815	**0.004863**	1.61

Table 7.3: Comparison of L_∞ and L_2 errors of original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the solution of the Buckley-Leverett equation with various parameters a and g, $T = 0.1$, $I = 128$.

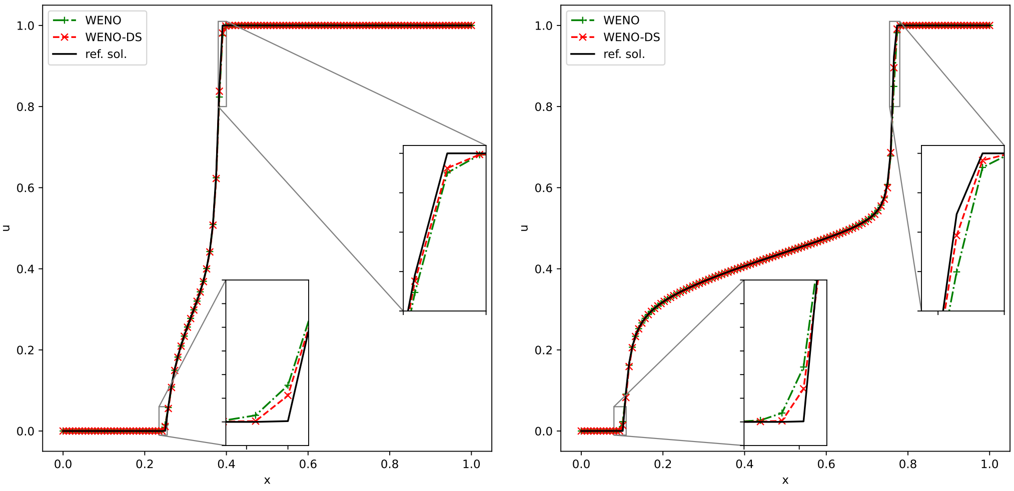
intervals. Some corresponding solutions can be found in Figures 7.8 and 7.9.

			L_∞			L_2		
T	a	g	WENO	WENO-DS	ratio	WENO	WENO-DS	ratio
0.05	1	5	0.060583	**0.017233**	3.52	0.006100	**0.003041**	2.01
	1	0	0.054741	**0.043503**	1.26	0.005102	**0.004072**	1.25
	0.5	2	0.024727	**0.018818**	1.31	0.003381	**0.002406**	1.41
	0.5	1	0.046306	**0.034351**	1.35	0.004535	**0.003500**	1.30
0.2	1	5	0.073220	**0.027178**	2.69	0.007083	**0.003397**	2.08
	1	0	0.040422	**0.027967**	1.45	0.003958	**0.002892**	1.37
	0.5	2	0.051640	**0.035332**	1.46	0.005300	**0.003984**	1.33
	0.5	1	0.039442	**0.030854**	1.28	0.004894	**0.004037**	1.21

Table 7.4: Comparison of L_∞ and L_2 errors of original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the solution of the Buckley-Leverett equation with various parameters a, g and T, $I = 128$.

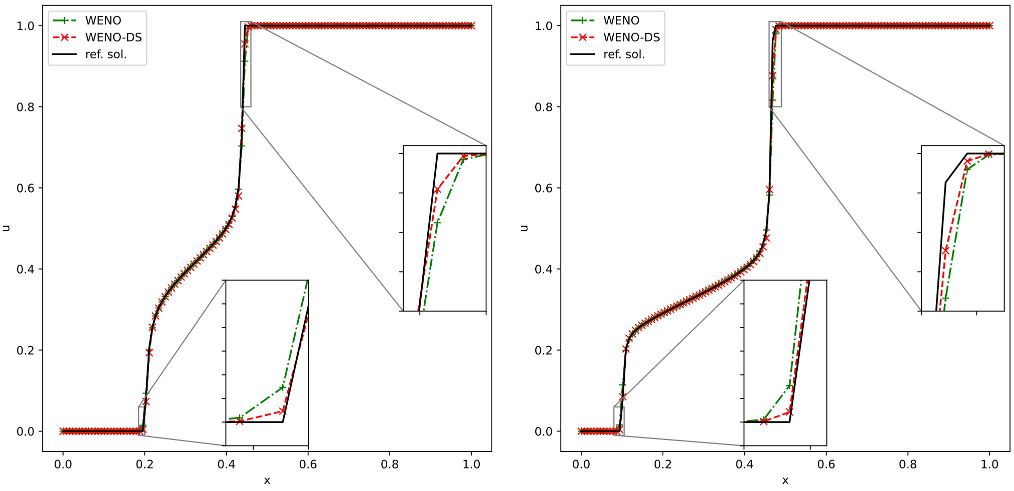
		L_∞			L_2		
a	g	WENO	WENO-DS	ratio	WENO	WENO-DS	ratio
0.5	9	0.060009	**0.050158**	1.20	0.007302	**0.006641**	1.10
0.75	6.5	0.087736	**0.045176**	1.94	0.008087	**0.005339**	1.51
0.25	7	0.146946	**0.086604**	1.70	0.013598	**0.007966**	1.71
0.6	8	0.075668	**0.064403**	1.17	**0.007357**	0.007401	0.99
0.25	9	0.168730	**0.102940**	1.64	0.016187	**0.010527**	1.54

Table 7.5: Comparison of L_∞ and L_2 errors of original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the solution of the Buckley-Leverett equation with various parameters a and g, $T = 0.05$, $I = 128$.



(a) $a = 0.5$, $g = 2$, $T = 0.05$ (b) $a = 1$, $g = 5$, $T = 0.2$

Figure 7.8: Comparison of the original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the numerical solution of the Buckley-Leverett equation with various parameters a, g and T, $I = 128$.



(a) $a = 0.75$, $g = 6.5$ (b) $a = 0.25$, $g = 7$

Figure 7.9: Comparison of the original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the numerical solution of the Buckley-Leverett equation with various parameters a and g, $T = 0.05$, $I = 128$.

7.6.4 The strongly degenerate parabolic convection-diffusion equation

In this example we test the method trained on the Buckley-Leverett data from Section 7.6.3. We do not retrain the method and apply it to the strongly degenerate parabolic convection-diffusion equation of the form specified by (7.62). Here we want to show the ability of WENO-DS to generalize on completely different and unseen problem. This is a benchmark example presented e.g. in [51, 63, 74]. We take $\epsilon = 0.1$, $f(u) = u^2$ and

$$\nu(u) = \begin{cases} 0, & |u| \leq 0.25, \\ 1, & |u| > 0.25. \end{cases} \tag{7.67}$$

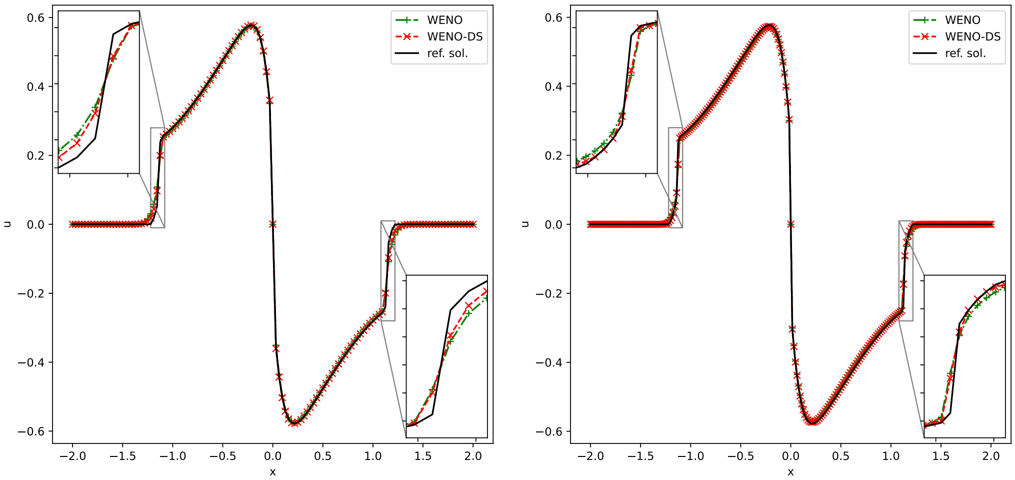
This leads to a fact, that the equation is hyperbolic if $u \in [-0.25, 0.25]$ and parabolic elsewhere. The parabolic term takes a form

$$b(u) = \begin{cases} \epsilon\,(u + 0,25), & u < -0.25, \\ \epsilon\,(u - 0,25), & u > 0.25, \\ 0, & u \leq |0.25|. \end{cases} \tag{7.68}$$

The initial condition is taken as

$$u(x,0) = \begin{cases} 1, & -\frac{1}{\sqrt{2}} - 0.4 < x \leq -\frac{1}{\sqrt{2}} + 0.4, \\ -1, & \frac{1}{\sqrt{2}} - 0.4 < x \leq \frac{1}{\sqrt{2}} + 0.4, \\ 0, & \text{otherwise}. \end{cases} \tag{7.69}$$

We use the zero Dirichlet boundary conditions and compute the solution to the final time $T = 0.7$ with $I = 128$, $I = 256$ and $I = 512$. We present the numerical results in Figure 7.10 and see that our method is able to capture the discontinuities and sharp interfaces very well. The reference solution is obtained using MWENO and WENO-Z method with $I = 1024$.



(a) $I = 128$ (b) $I = 256$

Figure 7.10: Numerical solution of the strongly degenerate parabolic equation, $T = 0.7$.

Moreover, we compare the errors in Table 7.6. Even that the method was trained on completely different training data and only with fixed discretization $I = 128$, we observe also for this example for all listed discretizations significantly smaller errors.

	L_∞			L_2		
I	WENO	WENO-DS	ratio	WENO	WENO-DS	ratio
128	0.055785	**0.044574**	1.25	0.022787	**0.017108**	1.33
256	0.070881	**0.061685**	1.15	0.015399	**0.012447**	1.24
512	0.042206	**0.019863**	2.12	0.006297	**0.004056**	1.55

Table 7.6: Comparison of L_∞ and L_2 errors of original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the solution of the strongly degenerate parabolic equation for various spatial discretizations, $T = 0.7$.

7.6.5 Two-dimensional porous medium equation with the Barenblatt solution

In the next example we solve the two-dimensional PME of the form

$$u_t = (u^m)_{xx} + (u^m)_{yy}, \qquad m > 1. \tag{7.70}$$

As an initial condition we use a Barenblatt solution (7.54) at time $t = 1$ with $d = 2$. In this case, the Barenblatt solution has no derivative at the points of the circle $x^2 + y^2 = \sqrt{\frac{4m}{\alpha(m-1)}}\, t^\alpha$, with $\alpha = \frac{1}{m}$. We choose the computational domain $\Omega = [-10, 10]$ and zero Dirichlet boundary conditions $u = 0$ on the boundary $\partial\Omega$. We divide the computational domain into 64×64 space grid points.

In our training we proceed analogously to the one-dimensional PME example and again simulate the equation (7.70) for $m \in \mathcal{U}(2, 8)$ to make the final numerical scheme more robust. We use the same CNN structure as described in Figure 7.1, the same loss function (7.56) and Adam optimizer with the learning rate 0.01 to update the weights. We show the progress of the validation metric function (7.57) in Figure 7.11 and see that the low values are obtained after a few first training steps. We choose the model obtained after the 500th training step, where we consider the method to be most effective.

Alternatively, we can use the method which was an output of the training procedure for the one-dimensional PME from the Section 7.6.1. We compare the errors of the both methods in Table 7.7. We see that the results are very similar and also the method trained on a one-dimensional example (and on one-dimensional data) can be reliably used in more-dimensional space. This observation can be very useful when the computation of a reference solution in more dimensions becomes too demanding. Figure 7.12 illustrates the solution for $m = 2$.

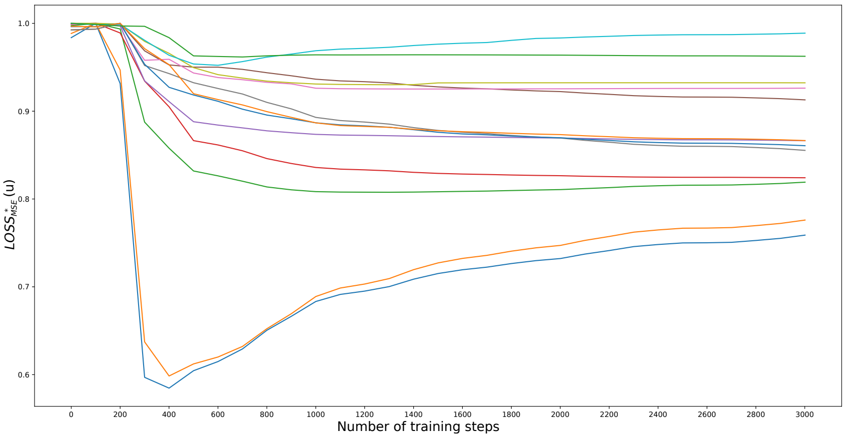


Figure 7.11: Training evolution corresponding to two-dimensional PME: The values (7.57) for different validation problems evaluated each 100 training steps.

	L_∞					L_2				
m	MWENO	WENO-DS (2d model)	ratio	WENO-DS (1d model)	ratio	MWENO	WENO-DS (2d model)	ratio	WENO-DS (1d model)	ratio
2	0.009582	0.008360	1.15	**0.008085**	1.19	0.000836	0.000666	1.26	**0.000653**	1.28
3	0.055924	**0.052861**	1.06	0.053215	1.05	0.004178	**0.003820**	1.09	0.003899	1.07
4	**0.102970**	0.104187	0.99	0.105067	0.98	0.009584	**0.009167**	1.05	0.009303	1.03
5	0.191146	**0.185514**	1.03	0.188885	1.01	0.015311	**0.014862**	1.03	0.015068	1.02
6	0.154870	**0.141617**	1.09	0.141799	1.09	0.012903	**0.012299**	1.05	0.012395	1.04
7	**0.268363**	0.269613	1.00	0.270895	0.99	0.019981	**0.019323**	1.03	0.019663	1.02
8	**0.298711**	0.299875	1.00	0.300904	0.99	0.021872	**0.021466**	1.02	0.021751	1.01

Table 7.7: Comparison of L_∞ and L_2 errors of MWENO and WENO-DS methods for the solution of the PME with various parameters m, $d = 2$, $T = 2$, 64×64 cells.

7.6.6 Interaction of two bumps for two-dimensional porous medium equation

In this test case, taken from [51, 74], we solve the two-dimensional PME

$$u_t = (u^2)_{xx} + (u^2)_{yy}, \tag{7.71}$$

with the initial condition

$$u(x, y, 0) = \begin{cases} \exp\left(\frac{-1}{6-(x-2)^2-(y+2)^2}\right), & (x-2)^2 + (y+2)^2 < 6, \\ \exp\left(\frac{-1}{6-(x+2)^2-(y-2)^2}\right), & (x+2)^2 + (y-2)^2 < 6, \\ 0, & \text{otherwise.} \end{cases} \tag{7.72}$$

We use the computational domain $\Omega = [-10, 10]$ and zero Dirichlet boundary conditions $u = 0$ on the boundary $\partial\Omega$.

We present the solution using the WENO-DS method from the Section 7.6.1. We

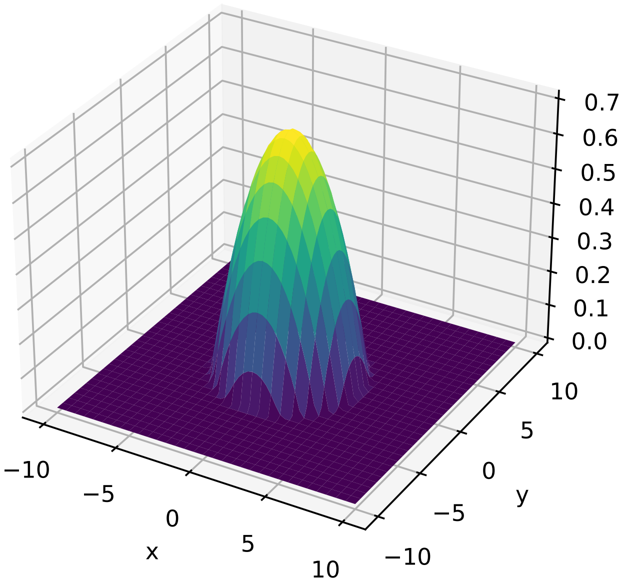


Figure 7.12: Numerical solution of the PME with $d = 2$, $m = 2$, $T = 2$, 64×64 cells.

did not retrain the CNN and applied the method directly. The solution for $T = 0$, 0.5, 1, 4 can be found in Figure 7.13. This agrees very well with the solution presented in [51, 74] and no noticeable oscillations are present.

Moreover, we compare the L_∞ and L_2 errors in Table 7.8. We compare the error values of a numerical solution computed on three different grid discretizations with a reference solution, which was computed on the spatial discretization with 512×512 cells using the MWENO method.

We used the method trained only on one-dimensional data with a Barenblatt solution and on fixed spatial discretization $I = 64$ and another ability of the WENO-DS scheme to generalize can be observed. In all presented cases we obtain improving results.

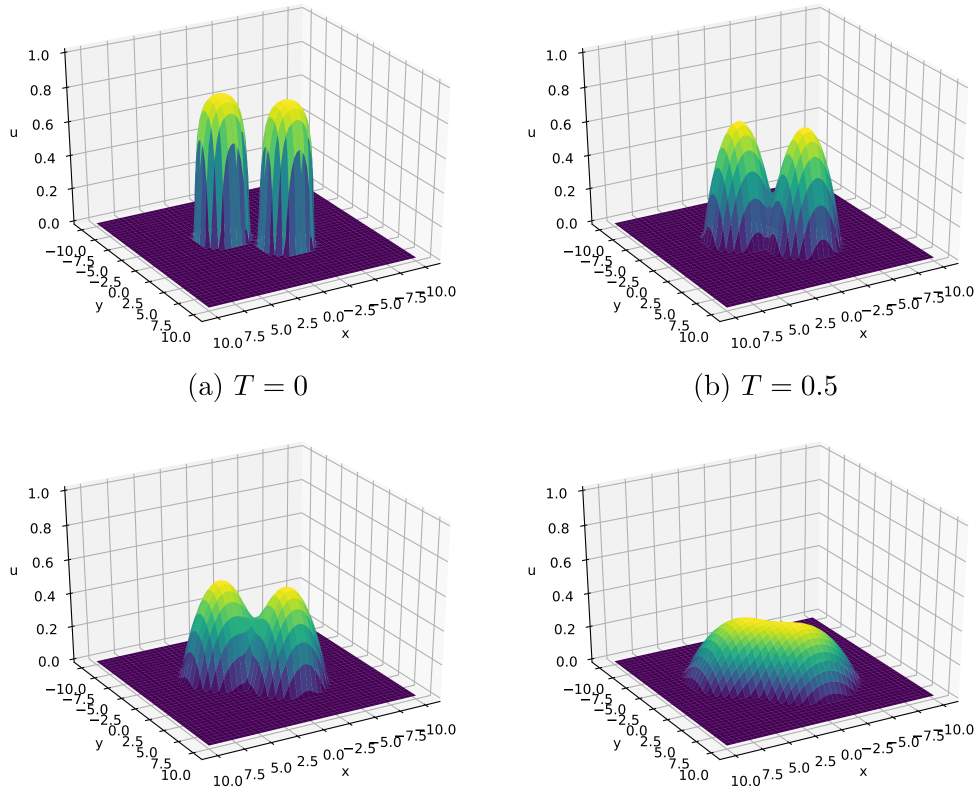
7.6.7 Two-dimensional Buckley-Leverett equation

As a last example we solve the two-dimensional Buckley-Leverett equation of the form

$$u_t + f_1(u)_x + f_2(u)_y = \epsilon\,(u_{xx} + u_{yy}), \tag{7.73}$$

with $\epsilon = 0.01$ and the flux functions

$$f_1(u) = \frac{u^2}{u^2 + (1-u)^2}, \qquad f_2(u) = f_1(u)\left(1 - 5(1-u)^2\right). \tag{7.74}$$



(a) $T = 0$

(b) $T = 0.5$

(c) $T = 1$

(d) $T = 4$

Figure 7.13: Numerical solution of the PME (7.71) with the initial condition (7.72), 64×64 cells.

We solve equation (7.73) with the WENO-DS method trained on the one-dimensional Buckley-Leverett equation from Section 7.6.3. We divide the computational domain $[-1.5, 1.5] \times [-1.5, 1.5]$ into 128×128 uniform cells and solve the equation with the initial condition

$$u(x, y, 0) = \begin{cases} 1, & x^2 + y^2 < 0.5, \\ 0, & \text{otherwise}. \end{cases} \tag{7.75}$$

The results at time $T = 0.5$ are presented in Figure 7.14 and agree with the results shown in [63]. With this example we demonstrate that the method trained on one-dimensional data can be easily applied also in more dimensions and provides a high quality numerical solution to the equation with a nonlinear, degenerate diffusion.

Also in this example, we computed the reference solution using standard WENO schemes on a grid with 512×512 cells. We compare the corresponding L_∞ and L_2 errors for different discretizations in Table 7.9. As can be seen, we obtain significant error improvements for all listed discretizations.

	L_∞			L_2		
T	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
0.5	0.013753	**0.010850**	1.27	0.001600	**0.001233**	1.30
1	0.011237	**0.009137**	1.23	0.001218	**0.000937**	1.30
4	0.004572	**0.003315**	1.38	0.000529	**0.000415**	1.27

(a) 64 × 64 cells.

	L_∞			L_2		
T	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
0.5	0.014062	**0.011446**	1.23	0.000435	**0.000313**	1.39
1	0.005134	**0.004037**	1.27	0.000311	**0.000222**	1.40
4	0.002036	**0.001585**	1.29	0.000135	**0.000101**	1.34

(b) 128 × 128 cells.

	L_∞			L_2		
T	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
0.5	0.004720	**0.003250**	1.45	0.000115	**0.000070**	1.64
1	0.001594	**0.001078**	1.48	0.000077	**0.000049**	1.56
4	0.000780	**0.000512**	1.52	0.000037	**0.000030**	1.25

(c) 256 × 256 cells.

Table 7.8: Comparison of L_∞ and L_2 errors of MWENO and WENO-DS methods for the solution of the PME (7.71) with the initial condition (7.72) and various parameters T.

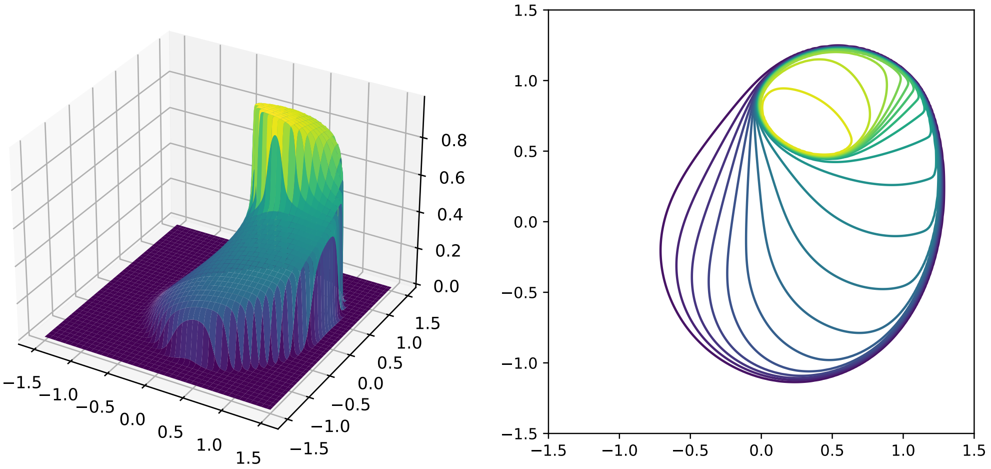


Figure 7.14: Numerical solution of the two-dimensional Buckley-Leverett equation at $T = 0.5$. 256 × 256 cells.

	L_∞			L_2		
$I \times I$	MWENO	WENO-DS	ratio	MWENO	WENO-DS	ratio
64×64	0.349060	**0.331053**	1.05	0.026759	**0.020641**	1.30
128×128	0.294226	**0.255652**	1.15	0.013842	**0.010647**	1.30
256×256	0.117042	**0.087451**	1.34	0.004047	**0.002192**	1.85

Table 7.9: Comparison of L_∞ and L_2 errors of MWENO and WENO-DS methods for the solution of the two dimensional Buckley-Leverett equation (7.73) with the initial condition (7.75) and various spatial discretizations.

8

Chapter 8

Deep smoothness WENO scheme with application in computational finance

The WENO scheme has broad applications not only in the area of hyperbolic conservation laws or nonlinear degenerate parabolic equations, but also in the area of the computational finance. In computational finance problems, we often face the problems with discontinuous initial or terminal data. Therefore, in 2015 the authors Hajipour and Malek applied the WENO scheme to the option pricing problems [36, 37]. The method was shown to be effective and non-oscillatory when solving the Black-Scholes equation. Further in [77], the application of the WENO method for the Asian option pricing problem can be found. A detailed study of application the WENO method in the area of computational finance was done in [56]. Here, not only option pricing problems, but also portfolio optimization problems were considered.

In this chapter we use the newly developed WENO-DS method for solving the Black-Scholes equation:

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV = 0, \quad t \in [0, T], \tag{8.1}$$

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.

Although the WENO scheme should avoid the spurious oscillations in the solution, they are still present in some cases, especially in the first time steps of the numerical solution. This motivates us to use the enhanced WENO-DS scheme presented in Chapters 5 and 7 for solving the European digital option pricing problem with the following terminal and boundary conditions:

$$\begin{gathered} V(S,T) = \begin{cases} 1, & \text{if} \quad S \geq K, \\ 0, & \text{if} \quad S < K, \end{cases} \\ V(S,t) \to 0, \quad \text{for} \quad S \to 0, \quad V(S,t) \to e^{-r(T-t)}, \quad \text{for} \quad S \to \infty, \end{gathered} \tag{8.2}$$

with K being a strike price.

We first use the following variable transformation:

$$S = Ke^x, \quad \tau = T - t, \quad V(S,t) = Ku(x,\tau), \tag{8.3}$$

and substitute this into (8.1) and (8.2). Then we obtain the (forward-in-time) PDE:

$$u_\tau = \frac{\sigma^2}{2} u_{xx} + \left(r - \frac{\sigma^2}{2}\right) u_x - ru, \quad x \in \mathbb{R},\ 0 \le \tau \le T, \tag{8.4}$$

with the transformed boundary conditions

$$u(x,t) = 0, \quad \text{for} \quad x \to 0, \quad u(x,t) = e^{-r\tau}/K, \quad \text{for} \quad x \to \infty. \tag{8.5}$$

8.1 Problem formulation

Let us consider the equation (8.4). This is a diffusion-convection-reaction PDE of the form

$$\frac{\partial u(x,\tau)}{\partial \tau} = a_0 \frac{\partial^2 u(x,\tau)}{\partial x^2} + a_1 \frac{\partial u(x,\tau)}{\partial x} + a_2 u(x,\tau), \quad (x,\tau) \in \Omega \times (0,\infty), \tag{8.6}$$

where a_0, a_1 and a_2 are constant coefficients. We introduce the uniform spatial grid defined by the points $x_i = x_0 + i\Delta x$, $i = 0, \dots, I$.

We consider the WENO discretization for diffusion and convection terms and we obtain the following semi-discrete formulation:

$$\frac{du_i(\tau)}{dt} = a_0 \frac{\hat{u}_{i+\frac{1}{2}} - \hat{u}_{i-\frac{1}{2}}}{\Delta x^2} + a_1 \frac{\tilde{u}_{i+\frac{1}{2}} - \tilde{u}_{i-\frac{1}{2}}}{\Delta x} + a_2 u_i(\tau), \quad \tau > 0, \tag{8.7}$$

where $u_i(\tau)$ approximates pointwise $u(x_i,\tau)$ and $\hat{u}_{i+1/2} = \hat{u}(u_{i-2}, \dots, u_{i+3})$, $\tilde{u}_{i+1/2} = \tilde{u}(u_{i-2}, \dots, u_{i+2})$ are the numerical flux functions. In order to obtain these values, the WENO-DS discretization is used. This is done as described in Chapters 5 and 7.

8.2 Application of deep learning and training procedure

To obtain the enhanced WENO-DS scheme for solving the European digital option pricing problem, we train a CNN on a large set of data. We use the CNN structure described in Figure 8.1. We emphasize that we use a very small CNN with the adaptive activation functions, as described in Chapter 6 in Section 6.1.1. We use the same CNN structure for training both WENO-DS for the hyperbolic term and WENO-DS for the parabolic term of (8.6).

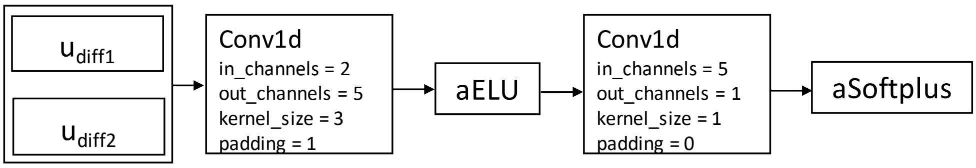


Figure 8.1: A structure of the CNN used for the European option pricing problem.

For the training, we set $K = 50$, $T = 1$, and randomly generate the parameters

$$\begin{aligned} \sigma &= 0.31 + \max(0.07a, -0.3), \\ r &= 0.11 + \max(0.07b, -0.1), \end{aligned} \tag{8.8}$$

where a and b are normally distributed. Here, the problems with different combinations of σ and r are covered. We use the computational domain $[x_L, x_R] = [-6, 1.5]$ partitioned into 100 space steps and use the temporal step size $\Delta\tau = 0.8\Delta x^2/\sigma^2$. As we mentioned earlier, the spurious oscillations mainly occur in the first time steps of a numerical solution. Therefore, we proceed with a training as follows.

First, the parameters (8.8) are randomly generated. We initialize the weights of the CNN randomly and perform a single time step of a solution. We compute the values u_{diff1}, u_{diff2}, which represent an effective preprocessing of the solution from the current time step, since they give us information about the smoothness of the solution. They are given by

$$u_{\text{diff1},i} = \bar{u}(\bar{x}_{i+1}) - \bar{u}(\bar{x}_{i-1}), \quad u_{\text{diff2},i} = \bar{u}(\bar{x}_{i+1}) - 2\bar{u}(\bar{x}_i) + \bar{u}(\bar{x}_{i-1}), \tag{8.9}$$

with

$$\begin{aligned} \bar{x}_i &= (x_{i-k}, x_{i-k+1}, \dots, x_{i+k}), \\ \bar{u}(\bar{x}_i) &= \big(u(x_{i-k}), u(x_{i-k+1}), \dots, u(x_{i+k})\big), \end{aligned} \tag{8.10}$$

where $2k+1$ is the size of the receptive field of the whole CNN. They are then used as input values for the first hidden layer. Next we calculate the loss defined by

$$\text{LOSS}(u) = \sum_{i=0}^{I-1} \big[\max(\hat{u}_i - \hat{u}_{i+1}, 0)\big], \tag{8.11}$$

where $\hat{u}_i$ is a numerical approximation of $u(x_i)$. This loss is positive, if the approximation of the solution is decreasing in x (in true solution it should be only increasing), so we test the monotonicity of the solution. After that, the gradient with respect to the weights of the CNN is calculated using the backpropagation algorithm. Then, the Adam optimizer [55] with a learning rate of 0.001 is used to update the weights. Next, we test the model on a validation set and repeat the above steps with newly generated parameters (8.8). We emphasize, that we train the CNN for randomly generated initial data but only on the first time step of a numerical solution. This is the solution which is most likely to obtain spurious oscillations, which we want to avoid. After the training, we select the weights from the training step, at which the model performed best on the validation problems.

It should be noted that for the temporal discretization, we use a third-order TVD Runge-Kutta method [100], imposing intermediate boundary conditions as in [37].

8.3 Numerical results

Let us present the numerical results of the training procedure described before. In Figure 8.2, we show the evolution of the loss values for the problems from the validation set. We rescale the values (8.11) to be in interval $[0, 1]$ using

$$\mathrm{LOSS}^*(u) = \frac{\mathrm{LOSS}^l(u)}{\max\limits_{l=0,\dots,L} (\mathrm{LOSS}^l(u))}, \qquad l = 0, \dots, L, \tag{8.12}$$

where L denotes the total number of training steps. We see that the loss is decreasing and select the model obtained after the last training step as our final WENO-DS scheme.

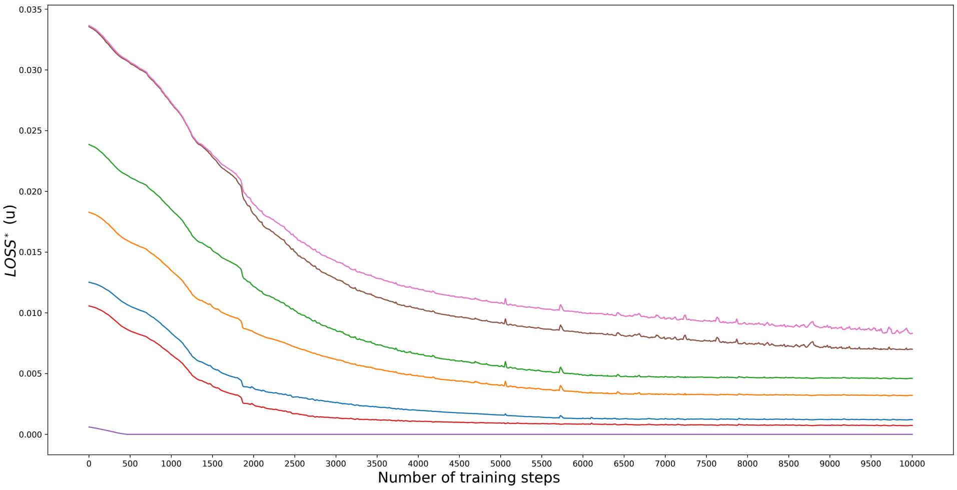
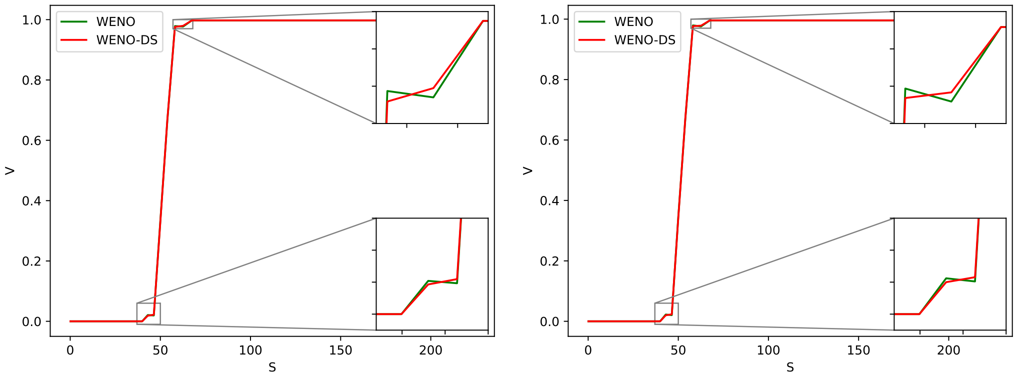


Figure 8.2: Training evolution corresponding to European digital option example: The values (8.12) for different validation problems.

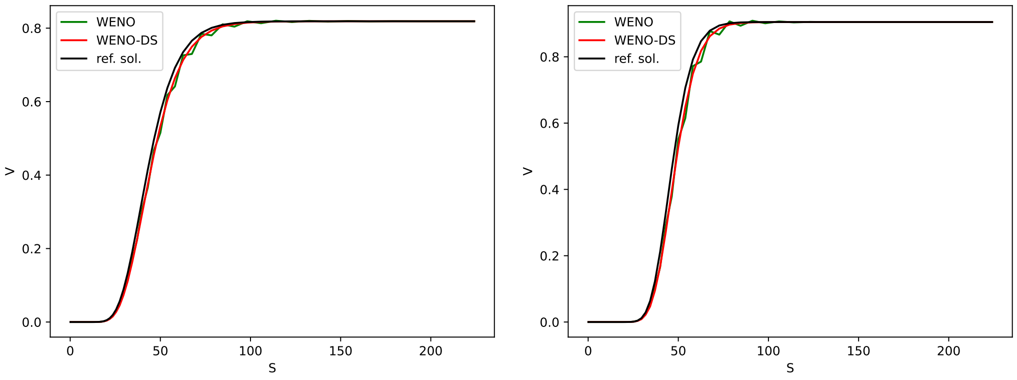
We compare the solution at the first time step in Figure 8.3 and see that the WENO-DS reliably eliminates the oscillations that occur when using the original WENO scheme (WENO-Z scheme [13] for the approximation of the hyperbolic term and MWENO scheme [35] for the approximation of the parabolic term).

In most cases, the original WENO scheme is able to handle these oscillations with increasing number of time steps. However, in some cases the oscillations are still present. Figure 8.4 shows the solution at time $T = 1$ and we see that our method produces a smooth solution in contrast to the original WENO method.

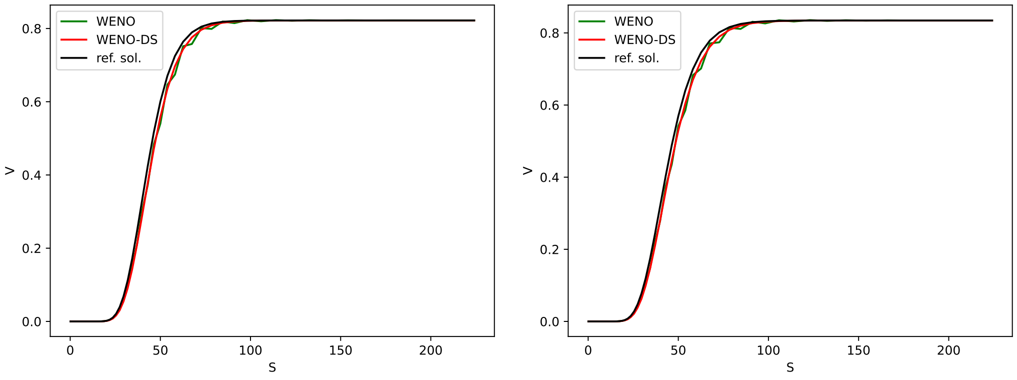


(a) $\sigma = 0.3$ and $r = 0.05$ (b) $\sigma = 0.4$ and $r = 0.15$

Figure 8.3: Comparison of the classical WENO (WENO-Z combined with MWENO) and WENO-DS methods, $I = 100$. Solution at the first time step.



(a) $\sigma = 0.3$ and $r = 0.2$ (b) $\sigma = 0.2$ and $r = 0.1$



(c) $\sigma = 0.263$ and $r = 0.196$ (d) $\sigma = 0.292$ and $r = 0.181$

Figure 8.4: Comparison of the classical WENO (WENO-Z combined with MWENO) and WENO-DS methods, $I = 100$. Solution at the last time step, $T = 1$.

We compare the L_∞ and L_2 errors in Table 8.1 and show that the WENO-DS method exhibits smaller errors in most cases. Thus, we are not only able to eliminate the spurious oscillations, but also improve the quality of the numerical solution.

		L_∞		L_2	
σ	r	WENO	WENO-DS	WENO	WENO-DS
0.28	0.13	0.000933	0.000934	0.000660	0.000660
0.1	0.05	0.002751	0.002753	0.001196	0.001193
0.3	0.2	0.001120	0.000811	0.000650	0.000594
0.2	0.1	0.001833	0.001335	0.000890	0.000803
0.15	0.05	0.002446	0.001823	0.001055	0.000971
0.4	0.1	0.000676	0.000676	0.000570	0.000569
0.3	0.05	0.000945	0.000945	0.000691	0.000691
0.4	0.15	0.000641	0.000641	0.000542	0.000542
0.263	0.196	0.001206	0.000926	0.000671	0.000637
0.292	0.181	0.001107	0.000853	0.000655	0.000614

Table 8.1: Comparison of the L_∞ and L_2 errors of original WENO (WENO-Z combined with MWENO) and WENO-DS methods for the solution of the transformed Black-Scholes equation (8.4) with various parameters σ and r, $I = 100$.

As we demonstrated in this section, WENO-DS significantly improve the quality of the numerical solutions, even in the cases, when the classical WENO scheme does not handle the oscillations properly.

9

Chapter 9

Conclusion and outlook

This thesis has been dedicated to the improvement of standard numerical schemes using deep learning. First, in Chapter 1, we have given the reader an overview of deep learning in order to better understand the techniques used in the rest of this thesis.

After providing detailed literature review of existing deep learning algorithms in numerical mathematics in Chapter 2, we developed a new deep learning based finite difference scheme for solving PDEs, which we call Deep FDM, in Chapter 3. 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 be easily extended to other numerical schemes, such as compact FDMs. The scheme is easy to use and provides improved numerical results, which are demonstrated on the examples presented. We have shown that the method generalizes well, i.e. it gives good results for parameters outside the training region, and remains time efficient even when the small neural network part is added.

The second part of this thesis includes a comprehensive study on the improvement of the WENO scheme using deep learning. In Chapter 4, we first introduced the hyperbolic conservation laws and the basic mathematical background.

In Chapter 5, we presented the methodology for improving the standard WENO schemes using deep learning. We first introduced the standard WENO schemes in more detail and then presented our approach. This is based on a modification of the smoothness indicators of the WENO method. To do this, we trained a relatively small neural network, but the smoothness indicators alone were not the output of the training procedure, but only the multiplicative perturbations of them. This ensures consistency of the scheme, and no additional post-processing is required. We have applied our improvement to the WENO-Z scheme, where the (formal) fifth-order accuracy of the smooth solutions can be proved analytically. Our new method, the WENO-DS scheme significantly improves the numerical results, especially in the presence of discontinuities, even for cases that have not been trained. We have demonstrated our results on the the Buckley-Leverett equation and inviscid Burgers equation. We have shown that the method can efficiently solve problems in more dimensional space without additional retraining.

Furthermore, in Chapter 6, we extended the previously discussed approach for solving one- and two-dimensional Euler systems of gas dynamics. We implemented numerous benchmark examples and demonstrated the superiority of WENO-DS over the standard WENO-Z scheme. In addition, a more effective training procedure is presented in this chapter.

In Chapter 7, we developed the WENO-DS scheme for the solution of nonlinear degenerate parabolic equations. Using deep learning techniques, we improved the smoothness indicators of the original WENO method and applied our improvement to the MWENO scheme. We preserved the sixth-order accuracy and proved it theoretically. On one- and two-dimensional benchmark examples from the literature, we showed that the WENO-DS method outperforms the MWENO scheme in the challenging examples of nonlinear degenerate parabolic equations, and remains sixth-order accurate in smooth regions of the domain.

Finally, in Chapter 8, we applied the WENO-DS method to computational finance problems, namely the European digital option pricing problem with discontinuous terminal data. In this problem, the spurious oscillations are present in the solution when the standard WENO schemes are used. We have shown that they can be successfully eliminated using the WENO-DS method.

9.1 Highlights of the thesis and outlook

To conclude this thesis, we would like to list its main highlights and present ideas for future research.

The main advantages of the developed Deep FDM scheme include

- The approach can be seen as a proof of concept that deep learning can be easily used to approximate the local discretization error of the numerical scheme for solving PDEs.
- The scheme is simple to use and straightforward to implement, making it easy to generalize to other standard numerical schemes.

In our future work, we will further investigate this approach by applying it to more innovative numerical schemes and to more challenging examples.

Key highlights of the next proposed WENO-DS schemes include

- By seamlessly integrating deep learning techniques into the WENO algorithm, we have successfully improved the accuracy of numerical solutions, especially in regions close to abrupt shocks.
- Unlike previous attempts to incorporate deep learning into numerical methods, this approach is unique in that it eliminates the need for additional post-processing steps and ensures consistency throughout. The corresponding proofs of the formal order of accuracy can be found in this thesis.
- The deep learning-based smoothness indicator in WENO-DS schemes preserves the conservative nature of the method, ensuring that WENO-DS pro-

duces the correct numerical solution.

- We have also demonstrated that the trained neural networks from one test case can be directly applied to another test case without any additional training. This demonstrates the effectiveness of the trained neural networks in dealing with unseen test cases. Such a strategy has the potential to increase the efficiency of integrating deep learning methods, especially for tackling large-scale computational problems.
- This study demonstrates the superiority of the WENO-DS approach by a comprehensive investigation of various test problems, including scenarios involving shocks and rarefaction waves. The results consistently show the enhanced capabilities of the approach, which surpass those of traditional fifth- and sixth-order WENO schemes, particularly in addressing challenges such as excessive diffusion or overshooting around shocks.

In summary, the WENO-DS approach represents a significant advance in the field of numerical methods for hyperbolic conservation laws. The incorporation of deep learning techniques has not only improved the accuracy, but also the qualitative behavior of the solutions, both in smooth regions and near discontinuities. This research paves the way for future developments at the intersection of traditional numerical methods and machine learning, and offers a promising direction for further progress in solving complex PDEs.

In future work, we will extend our investigation to a wider range of test cases, including complex flow patterns and diverse boundary conditions. We intend to include extended test cases that include the investigation of complex shock interactions, turbulence and a variety of boundary conditions. In addition, we will perform scalability studies to assess how the approach behaves as the problem size increases. These investigations aim to deepen our understanding of the robustness and generalizability of the WENO-DS approach, thereby strengthening its position in the field of numerical solution methods.

References

[1] R. Abedian. A new high-order weighted essentially non-oscillatory scheme for non-linear degenerate parabolic equations. *Numerical Methods for Partial Differential Equations*, 37(2):1317–1343, 2021.

[2] R. Abedian, H. Adibi, and M. Dehghan. A high-order weighted essentially non-oscillatory (WENO) finite difference scheme for nonlinear degenerate parabolic equations. *Computer Physics Communications*, 184(8):1874–1888, 2013.

[3] H. W. Alt and S. Luckhaus. Quasilinear elliptic-parabolic differential equations. *Mathematische Zeitschrift*, 183(3):311–341, 1983.

[4] F. Aràndiga, A. Baeza, A. M. Belda, and P. Mulet. Analysis of WENO schemes for full and global accuracy. *SIAM Journal on Numerical Analysis*, 49(2):893–915, 2011.

[5] A. Araujo, W. Norris, and J. Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019.

[6] D. G. Aronson. The porous medium equation. In *Nonlinear Diffusion Problems*, pages 1–46. Springer, 1986.

[7] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[8] G. I. Barenblatt. On self-similar motions of a compressible fluid in a porous medium. *Academy of Sciences of the USSR. Applied Mathematics and Mechanics*, 16(6), 1952.

[9] A. D. Beck, J. Zeifang, A. Schwarz, and D. Flad. A neural network based shock detection and localization approach for discontinuous Galerkin methods. *Journal of Computational Physics*, 423:109824, 2020.

[10] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck. An overview on deep learning-based approximation methods for partial differential equations. *Discrete and Continuous Dynamical Systems-B*, 28:3697–3746, 2023.

[11] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[12] J. Blechschmidt and O. G. Ernst. Three ways to solve partial differential equations with neural networks – a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

[13] R. Borges, M. Carmona, B. Costa, and W. S. Don. An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws. *Journal of Computational Physics*, 227(6):3191–3211, 2008.

[14] W. Cao, Q. Xu, and Z. Zheng. Solution of two-dimensional time-fractional Burgers equation with high and low Reynolds numbers. *Advances in Difference Equations*, 2017(1):1–14, 2017.

[15] M. Castro, B. Costa, and W. S. Don. High order weighted essentially non-oscillatory WENO-Z schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 230(5):1766–1792, 2011.

[16] T. Chang, G.-Q. Chen, and S. Yang. On the 2-D Riemann problem for the compressible Euler equations I. Interaction of shocks and rarefaction waves. *Discrete and Continuous Dynamical Systems*, 1(4):555–584, 1995.

[17] T. Chang, G.-Q. Chen, and S. Yang. On the 2-D Riemann problem for the compressible Euler equations II. Interaction of contact discontinuities. *Discrete and Continuous Dynamical Systems*, 6(2):419–430, 1999.

[18] A. Christlieb, W. Guo, Y. Jiang, et al. Kernel based high order "explicit" unconditionally stable scheme for nonlinear degenerate advection-diffusion equations. *Journal of Scientific Computing*, 82(3):52, 2020.

[19] M. G. Crandall and A. Majda. Monotone difference approximations for scalar conservation laws. *Mathematics of Computation*, 34:1–21, 1980.

[20] N. Discacciati, J. S. Hesthaven, and D. Ray. Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. *Journal of Computational Physics*, 409:109304, 2020.

[21] W.-S. Don and R. Borges. Accuracy of the weighted essentially non-oscillatory conservative finite difference schemes. *Journal of Computational Physics*, 250:347–372, 2013.

[22] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[23] W. E and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[24] L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. Rhode Island, USA, 1998.

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

[26] Y. Feng, T. Liu, and K. Wang. A characteristic-featured shock wave indicator for conservation laws based on training an artificial neuron. *Journal of Scientific Computing*, 83(1):1–34, 2020.

[27] J. Fernández-Fidalgo, L. Ramírez, P. Tsoutsanis, I. Colominas, and X. Nogueira. A reduced-dissipation WENO scheme with automatic dissipation adjustment. *Journal of Computational Physics*, 425:109749, 2021.

[28] L. Fu. A hybrid method with TENO based discontinuity indicator for hyperbolic conservation laws. *Communications in Computational Physics*, 26(4):973–1007, 2019.

[29] L. Fu, X. Y. Hu, and N. A. Adams. A family of high-order targeted ENO schemes for compressible-fluid simulations. *Journal of Computational Physics*, 305:333–359, 2016.

[30] L. Fu, X. Y. Hu, and N. A. Adams. A new class of adaptive high-order targeted ENO schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 374:724–751, 2018.

[31] S. K. Godunov. *Different Methods For Shock Waves.* PhD thesis, Moscow State University, 1954.

[32] S. K. Godunov. A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations. *Matematicheskii Sbornik*, 47:271–306, 1959.

[33] A. Griewank et al. On automatic differentiation. *Mathematical Programming: Recent Developments and Applications*, 6:83–107, 1989.

[34] Y. Ha, C. H. Kim, Y. J. Lee, and J. Yoon. An improved weighted essentially non-oscillatory scheme with a new smoothness indicator. *Journal of Computational Physics*, 232(1):68–86, 2013.

[35] M. Hajipour and A. Malek. High accurate NRK and MWENO scheme for nonlinear degenerate parabolic PDEs. *Applied Mathematical Modelling*, 36(9):4439–4451, 2012.

[36] M. Hajipour and A. Malek. Efficient high-order numerical methods for pricing of options. *Computational Economics*, 45(1):31–47, 2015.

[37] M. Hajipour and A. Malek. High accurate modified WENO method for the solution of Black–Scholes equation. *Computational and Applied Mathematics*, 34(1):125–140, 2015.

[38] A. Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49(3):357–393, 1983.

[39] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. In *Upwind and High-Resolution Schemes*, pages 218–290. Springer, 1987.

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

[41] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[42] A. K. Henrick, T. D. Aslam, and J. M. Powers. Mapped weighted essentially non-oscillatory schemes: achieving optimal order near critical points. *Journal of Computational Physics*, 207(2):542–567, 2005.

[43] D. J. Hill and D. I. Pullin. Hybrid tuned center-difference-WENO method for large eddy simulations in the presence of strong shocks. *Journal of Computational Physics*, 194(2):435–450, 2004.

[44] G. Hinton, N. Srivastava, and K. Swersky. Lecture 6-RMSProp, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

[45] J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, and S. Ermon. Learning neural PDE solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

[46] A. D. Jagtap and G. E. Karniadakis. How important are activation functions in regression and classification? A survey, performance comparison, and future directions. *Journal of Machine Learning for Modeling and Computing*, 4(1):21–75, 2023.

[47] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[48] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476(2239):20200334, 2020.

[49] A. D. Jagtap, Y. Shin, K. Kawaguchi, and G. E. Karniadakis. Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing*, 468:165–180, 2022.

[50] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, 126(1):202–228, 1996.

[51] Y. Jiang. High order finite difference multi-resolution WENO method for nonlinear degenerate parabolic equations. *Journal of Scientific Computing*, 86(1):1–20, 2021.

[52] J. C. S. Kadupitiya, G. C. Fox, and V. Jadhao. Solving Newton's equations of motion with large time steps using recurrent neural networks based operators. *Machine Learning: Science and Technology*, 3(2):025002, 2022.

[53] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[54] C. H. Kim, Y. Ha, and J. Yoon. Modified non-linear weights for fifth-order weighted essentially non-oscillatory schemes. *Journal of Scientific Computing*, 67(1):299–323, 2016.

[55] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Published as a conference paper at ICLR 2015.

[56] T. Kossaczká. The weighted essentially non-oscillatory method for problems in finance. Master's thesis, Bergische Universität Wuppertal, Germany, 2019.

[57] T. Kossaczká, M. Ehrhardt, and M. Günther. Enhanced fifth order WENO shock-capturing schemes with deep learning. *Results in Applied Mathematics*, 12:100201, 2021.

[58] T. Kossaczká, M. Ehrhardt, and M. Günther. A deep smoothness WENO method with applications in option pricing. In *Progress in Industrial Mathematics at ECMI 2021*, pages 417–423. Springer, 2022.

[59] T. Kossaczká, M. Ehrhardt, and M. Günther. A neural network enhanced weighted essentially non-oscillatory method for nonlinear degenerate parabolic equations. *Physics of Fluids*, 34(2):026604, 2022.

[60] T. Kossaczká, M. Ehrhardt, and M. Günther. Deep FDM: Enhanced finite difference methods by deep learning. *Franklin Open*, 4:100039, 2023.

[61] T. Kossaczká, M. Ehrhardt, and M. Günther. Deep finite difference method for solving Asian option pricing problems. To appear in *Progress in Industrial Mathematics at ECMI 2023*, 2024.

[62] T. Kossaczká, A. D. Jagtap, and M. Ehrhardt. Deep smoothness weighted essentially non-oscillatory method for two-dimensional hyperbolic conservation laws: A deep learning approach for learning smoothness indicators. *Physics of Fluids*, 36(3):036603, 2024.

[63] A. Kurganov and E. Tadmor. New high-resolution central schemes for nonlinear conservation laws and convection-diffusion equations. *Journal of Computational Physics*, 160(1):241–282, 2000.

[64] A. Kurganov and E. Tadmor. Solution of two-dimensional Riemann problems for gas dynamics without Riemann problem solvers. *Numerical Methods for Partial Differential Equations*, 18(5):584–608, 2002.

[65] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[66] P. Lax and B. Wendroff. Systems of conservation laws. *Communications on Pure and Applied Mathematics*, 13(2):217–237, 1960.

[67] P. D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics*, 7(1):159–193, 1954.

[68] S. K. Lele. Compact finite difference schemes with spectral-like resolution. *Journal of Computational Physics*, 103(1):16–42, 1992.

[69] R. J. LeVeque. *Numerical methods for conservation laws*, volume 214. Springer, 1992.

[70] R. J. LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge University Press, 2002.

[71] L. Li, H. B. Wang, G. Y. Zhao, et al. Efficient WENOCU4 scheme with three different adaptive switches. *Journal of Zhejiang University: Science A*, 21(9):695–720, 2020.

[72] Q. Liu and X. Wen. The WENO reconstruction based on the artificial neural network. *Advances in Applied Mathematics*, 9(4):574–583, 2020.

[73] Y. Liu. Globally optimal finite-difference schemes based on least squares. *Geophysics*, 78(4):T113–T132, 2013.

[74] Y. Liu, C.-W. Shu, and M. Zhang. High order finite difference WENO schemes for nonlinear degenerate parabolic equations. *SIAM Journal on Scientific Computing*, 33(2):939–965, 2011.

[75] Y. Lu and W. Jäger. On solutions to nonlinear reaction–diffusion–convection equations with degenerate diffusion. *Journal of Differential Equations*, 170(1):1–21, 2001.

[76] Y. L. Ming et al. Deep Nitsche Method: Deep Ritz Method with essential boundary conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2021.

[77] C. W. Oosterlee, J. C. Frisch, and F. J. Gaspar. TVD, WENO and blended BDF discretizations for Asian options. *Computing and Visualization in Science*, 6(2-3):131–138, 2004.

[78] F. Otto. L^1-contraction and uniqueness for quasilinear elliptic-parabolic equations. *Journal of Differential Equations*, 131(1):20–38, 1996.

[79] A. Paszke et al. PyTorch: An imperative style, high-performance deep learn-

ing library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035. Curran Associates, Inc., 2019.

[80] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[81] S. Pirozzoli. Conservative hybrid compact-WENO schemes for shock-turbulence interaction. *Journal of Computational Physics*, 178(1):81–117, 2002.

[82] J. Qiu and C.-W. Shu. Hermite WENO schemes and their application as limiters for Runge-Kutta discontinuous Galerkin method: One-dimensional case. *Journal of Computational Physics*, 193(1):115–135, 2004.

[83] J. Qiu and C.-W. Shu. Hermite WENO schemes and their application as limiters for Runge-Kutta discontinuous Galerkin method II: Two dimensional case. *Computers & Fluids*, 34(6):642–663, 2005.

[84] M. Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[85] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[86] A. A. Ramabathiran and P. Ramachandran. SPINN: sparse, physics-based, and partially interpretable neural networks for PDEs. *Journal of Computational Physics*, 445:110600, 2021.

[87] S. Rathan, R. Kumar, and A. D. Jagtap. L^1-type smoothness indicators based WENO scheme for nonlinear degenerate parabolic equations. *Applied Mathematics and Computation*, 375:125112, 2020.

[88] S. Rathan and G. N. Raju. A modified fifth-order WENO scheme for hyperbolic conservation laws. *Computers & Mathematics with Applications*, 75(5):1531–1549, 2018.

[89] D. Ray and J. S. Hesthaven. Detecting troubled-cells on two-dimensional unstructured grids using a neural network. *Journal of Computational Physics*, 397:108845, 2019.

[90] L. C. G. Rogers and Z. Shi. The value of an Asian option. *Journal of Applied Probability*, 32(4):1077–1088, 1995.

[91] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory, 1957.

[92] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation, 1985.

[93] S. Saha et al. Deep Learning Discrete Calculus (DLDC): a family of discrete numerical methods by universal approximation for STEM education to frontier research. *Computational Mechanics*, 72:311–331, 2023.

[94] C. W. Schulz-Rinne. Classification of the Riemann problem for two-dimensional gas dynamics. *SIAM Journal on Mathematical Analysis*, 24(1):76–88, 1993.

[95] C. W. Schulz-Rinne, J. P. Collins, and H. M. Glaz. Numerical solution of the Riemann problem for two-dimensional gas dynamics. *SIAM Journal on Scientific Computing*, 14(6):1394–1414, 1993.

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

[97] J. Shi, C. Hu, and C.-W. Shu. A technique of treating negative weights in WENO schemes. *Journal of Computational Physics*, 175(1):108–127, 2002.

[98] C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations. Lecture Notes in Mathematics*, volume 1697. Springer, Berlin, 1998.

[99] C.-W. Shu. High order weighted essentially nonoscillatory schemes for convection dominated problems. *SIAM Review*, 51(1):82–126, 2009.

[100] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, 1988.

[101] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. In *Upwind and High-Resolution Schemes*, pages 328–374. Springer, 1989.

[102] C.-W. Shu, T. A. Zang, G. Erlebacher, D. Whitaker, and S. Osher. High-order ENO schemes applied to two- and three-dimensional compressible flow. *Applied Numerical Mathematics*, 9(1):45–71, 1992.

[103] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[104] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, 1978.

[105] B. Stevens and T. Colonius. Enhancement of shock-capturing methods via machine learning. *Theoretical and Computational Fluid Dynamics*, 34(3):483–496, 2020.

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

[107] C. K. Tam and J. C. Webb. Dispersion-relation-preserving finite difference schemes for computational acoustics. *Journal of Computational Physics*, 107(2):262–281, 1993.

[108] N. Trask, R. G. Patel, B. J. Gross, and P. J. Atzberger. GMLS-Nets: A framework for learning from unstructured data. *arXiv preprint arXiv:1909.05371*, 2019.

[109] C. J. Van Duyn and L. A. Peletier. Nonstationary filtration in partially saturated porous media. *Archive for Rational Mechanics and Analysis*, 78(2):173–198, 1982.

[110] B. Van Leer. Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme. *Journal of Computational Physics*, 14(4):361–370, 1974.

[111] J. L. Vázquez. *The Porous Medium Equation: Mathematical Theory.* Oxford University Press, 2007.

[112] R. Wang and R. J. Spiteri. Linear instability of the fifth-order WENO method. *SIAM Journal on Numerical Analysis*, 45(5):1871–1901, 2007.

[113] Y. Wang, Z. Shen, Z. Long, and B. Dong. Learning to discretize: Solving 1D scalar conservation laws via deep reinforcement learning. *Communications in Computational Physics*, 28(5):2158–2179, 2020.

[114] Z. J. Wang and R. F. Chen. Optimized weighted essentially nonoscillatory schemes for linear waves with discontinuity. *Journal of Computational Physics*, 174(1):381–404, 2001.

[115] P. Wesseling. *Principles of Computational Fluid Dynamics*, volume 29. Springer Science & Business Media, 2009.

[116] Y. B. Zel'dovich and A. S. Kompaneets. Towards a theory of heat conduction with thermal conductivity depending on the temperature. *Collection of papers dedicated to 70th birthday of Academician A. F. Ioffe, Izd. Akad. Nauk SSSR, Moscow*, pages 61–71, 1950.

[117] T. Zhang and Y. X. Zheng. Conjecture on the structure of solutions of the Riemann problem for two-dimensional gas dynamics systems. *SIAM Journal on Mathematical Analysis*, 21(3):593–630, 1990.

[118] F. Zhao, X. Chen, J. Wang, Z. Shi, and S. L. Huang. Performance-guaranteed ODE solvers with Complexity-Informed Neural Networks. In *The Symbiosis of Deep Learning and Differential Equations*, pages 1–6, 2021.

[119] Z. Zhao, Y. T. Zhang, Y. Chen, and J. Qiu. A Hermite WENO method with modified ghost fluid method for compressible two-medium flow problems. *Communications in Computational Physics*, 30(3):851–873, 2021.