On the Approximation of Rough Functions with Artificial Neural Networks

Master Thesis

Author(s):
De Ryck, Tim

Publication date:
2020-01-31

Permanent link:
https://doi.org/10.3929/ethz-b-000397533

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

On the Approximation of Rough Functions with Artificial Neural Networks

Master's Thesis

Tim De Ryck

Friday 31st January, 2020

Advisor: Prof. Dr. Siddhartha Mishra
Department of Mathematics, ETH Zürich

Abstract

Deep neural networks and the ENO procedure are both efficient frameworks for approximating rough functions. We prove that at any order, the stencil shifts of the ENO and ENO-SR interpolation procedures can be exactly obtained using a deep ReLU neural network. In addition, we construct and provide error bounds for ReLU neural networks that directly approximate the output of the ENO and ENO-SR interpolation procedures. This surprising fact enables the transfer of several desirable properties of the ENO procedure to deep neural networks, including its high-order accuracy at approximating Lipschitz functions. Numerical tests for the resulting neural networks show excellent performance for interpolating rough functions, data compression and approximating solutions of nonlinear conservation laws.

Acknowledgements

In the first place, I would like to thank my supervisor Prof. Dr. Siddhartha Mishra. He suggested a topic that seamlessly combines two of my research interests, numerical mathematics and deep learning. This, together with his excellent guidance, made it a pleasure to develop this thesis. In particular, I thank him for providing me with a workplace at SAM and for giving me the chance to work on a research paper.

Second, I owe thanks to Dr. Deep Ray for the pleasant collaboration on this topic. It is his research findings that constitute the foundation of this thesis. I am particularly grateful to him for letting me use his code and for the fruitful discussion on ENO regression networks.

Finally, I am thankful for my parents and the opportunities they provided. My time in Zürich has been a great experience and it would not have been possible without their support.

Contents

Chapter 1

Introduction

Functions of limited regularity arise in a wide variety of problems in mathematics, physics and engineering. Weak solutions of nonlinear partial differential equations constitute a prominent class of such rough functions. For instance, solutions of Hamilton-Jacobi equations and other fully nonlinear PDEs are in general only Lipschitz continuous [Evans, 1998]. Discontinuous functions appear as solutions of nonlinear hyperbolic systems of conservation laws, e.g. the compressible Euler equations of gas dynamics, as they can contain shock waves [Dafermos, 2010]. Similarly, solutions to the incompressible Euler equations would well be only Hölder continuous in the turbulent regime [Eyink and Sreenivasan, 2006]. Another notable class of rough functions are images. On account of their sharp edges, they are usually assumed to be no more than functions of bounded variation.

In view of their prevalence, the need for numerical procedures that can efficiently and robustly approximate rough functions can not be undervalued. However, in classical approximation theory the accuracy of the interpolant usually relies on the regularity of the underlying function in a critical way. This poses a severe problem for the interpolation (or approximation) of rough functions. For continuous, piecewise smooth functions, standard linear interpolation procedures are only first-order accurate [Aràndiga et al., 2005] and the order of accuracy (in terms of the interpolation mesh width) degrades even further if the underlying function is discontinuous. Apart from a reduced accuracy, classical approximation procedures can also cause the emergence of spurious oscillations near jump discontinuities. This issue particularly arises when dealing with solutions of nonlinear hyperbolic systems of conservation laws that contain shock waves. The approximation of rough functions clearly poses a formidable challenge.

In Chapter 2 of this thesis, we present the concept of artificial neural networks. Inspired by the neural circuits in the human brain, they are formed by concatenating affine transformations with pointwise application of nonlinearities, the so-called activation functions. Being only piecewise smooth themselves, interpolation using neural networks will lead to significantly different results than the polynomial approximations arising from classical interpolation theory. The expressiveness of neural networks manifests itself through the universal approximation properties the networks possess, as will be discussed in Chapter 3. In particular, they can approximate piecewise smooth functions with an arbitrary degree of accuracy. These universality results however do not specify the precise architecture of the approximating network. This changed recently, when for example [Yarotsky, 2017] was able to explicitly construct deep neural networks with ReLU activation functions that can approximate Lipschitz functions to second-order accuracy. Moreover, [Yarotsky and Zhevnerchuk, 2019] were able to construct deep neural networks with alternating ReLU and sine activation functions that can approximate Lipschitz (or Hölder continuous) functions to exponential accuracy. In both cases, very precise estimates on the type and architecture of the underlying networks were given. These results give insight in how artificial neural networks can be used to construct approximation of rough functions and are therefore a starting point for this thesis.

Although the importance of the results of Yarotsky is well established, as they illustrate the power of deep neural networks in approximating rough functions, there is a practical issue in the use of these deep neural networks we want to address. The neural networks constructed in [Yarotsky, 2017] are mappings f_N from the space coordinate $x \in D \subset \mathbb{R}^d$ to the output $f_N(x) \in \mathbb{R}$, with as goal to approximate an underlying function $f : D \to \mathbb{R}$. Hence, for every given function f, a new neural network f_N has to be obtained. This happens through the minimization of a loss function with respect to a finite set of samples of f [Goodfellow et al., 2016]. Although it makes sense to train networks for each individual function f in high dimensions, for instance in the context of uncertainty quantification of PDEs [Lye et al., 2019], doing so for every low-dimensional function is unrealistic as this minimization procedure is computationally far more expensive than classical interpolation procedures. This problem can be resolved by noting that in a large numbers of contexts, the goal of approximating a function f is in fact finding a mapping $\{f(x_i)\}_i \mapsto \mathcal{I}f$, where $\{f(x_i)\}_i$ is a vector containing the function values of f at some sampling points and where the interpolant $\mathcal{I}f$ approximates f. Hence, one would like to construct neural networks that map the input vector to an output interpolant (or its evaluation at certain sampling points). It is unclear if the networks proposed by Yarotsky can be adapted to this setting.

Another (a priori very different) way to interpolate rough functions with a high accuracy is the use of data dependent interpolation procedures, a renowned example being the essentially non-oscillatory (ENO) procedure, which we will introduce in Chapter 4. It was first developed in the context of the reconstruction of non-oscillatory polynomials from cell averages and has proven its value in high-order accurate finite volume schemes for the approximation of the solutions of hyperbolic PDEs [Harten et al., 1987]. Moreover, ENO was shown to satisfy a subtle non-linear stability property, the so-called *sign property* [Fjordholm et al., 2013]. Later, ENO was adapted for interpolating rough functions [Shu and Osher, 1989]. Once augmented with a sub-cell resolution (SR) procedure of [Harten, 1989], which will be introduced in Chapter 5, it was provided in [Aràndiga et al., 2005] that the ENO-SR interpolant also approximated (univariate) Lipschitz functions to second-order accuracy. On account of their desirable properties, the ENO and ENO-SR interpolation procedures have also been successfully employed in the numerical approximation of Hamilton-Jacobi equations [Shu and Osher, 1991] and in data compression in image processing [Harten et al., 1997, Aràndiga et al., 2005].

As both deep neural networks and the ENO procedure provide frameworks for the efficient approximation of rough functions, it is natural to explore links between these two a priori disconnected concepts. The main goal of this thesis is to shed light on these links. In Chapter 4, we argue that the ENO(-SR) interpolation procedure can either be interpreted as a classification or a regression problem. We prove that for any order, the ENO interpolation procedure can be cast as a suitable deep ReLU neural network in the classification context. When interpreted as a regression problem, we prove the existence of a deep ReLU neural network that approximates the ENO interpolation procedure without being deprived of some of ENO's desirable properties. In Chapter 5, we propose a new variant of the piecewise linear ENO-SR procedure of [Harten, 1989] and prove again that it can be cast as a deep ReLU neural network in the classification context. In addition, we prove that there exists a deep ReLU neural network that provides an approximation of a piecewise smooth function that is second-order accurate, up to an arbitrarily small constant error.

In Chapter 6, we investigate whether it is possible to (re)obtain (or *train*) ReLU neural networks that satisfy the accuracy results of Chapter 4 and Chapter 5 based on a so-called training data set. This data set contains vectors of function evaluations on a grid and their corresponding ENO(-SR) interpolants. We describe in detail how to generate these data sets and how to train networks to obtain what we term as DeLENO (Deep Learning ENO) approximation procedures for rough functions. Furthermore, we test the performance of these newly proposed DeLENO procedures in different contexts and explore the effect of input scaling on the performance. To stress their practicality, we conclude the thesis by showing how the DeLENO interpolation procedures can be applied for function approximation, data compression and the approximation of solutions of conservation laws.

Chapter 2

Preliminaries

In statistics, machine learning, numerical mathematics and many other scientific disciplines, the goal of a certain task can often be reduced to the following. We consider a (usually unknown) function $f : D \subset \mathbb{R}^m \to \mathbb{R}^n$ and we assume access to a (finite) set of labelled data $\mathbb{S} \subset \{(X, f(X)) : X \in D\}$, using which we wish to select an approximation $\hat{f}$ from a parametrized function class $\{f^\theta : \theta \in \Theta\}$ that predicts the outputs of f on D with a high degree of accuracy. Classical choices include affine functions (linear regression), splines and many others. In the past decade, the use of artificial neural networks has gained popularity. In the following section, this class of functions is introduced, with special attention for ReLU deep neural networks, as these will form the cornerstone of this thesis. Afterwards, we present some very basic properties of ReLU neural networks.

2.1 Feedforward artificial neural networks

Artificial neural networks (ANNs) can be divided into many different categories. One of the most straightforward network types is the multilayer perceptron (MLP) or feedforward artificial neural network. These models are called feedforward because of the absence of feedback connections. Artificial neural networks that do have such connections are called recurrent neural networks. In feedforward artificial neural networks, a feedforward structure is created by stacking neurons, which are the basic computing units, in multiple layers. The input is fed into the *source layer* and flows through a number of *hidden layers* to the *output layer*. An example of an MLP with two hidden layers is shown in Figure 2.1.

In the following, we formalize the definition of a feedforward artificial neural network.

Definition 2.1 *Let $L, n_0, n_1, \ldots, n_L \in \mathbb{N}$ with $L \geqslant 2$, let $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $b^l \in \mathbb{R}^{n_l}$ for $l = 1, 2 \ldots, L$. Let $\mathcal{A}^l$ be the affine linear map given by $\mathcal{A}^l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l} : x \mapsto W^l x + b^l$ and let $\rho^l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ be a function, for each $l = 1, 2 \ldots, L$. A map $\Phi : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ defined by*

$$\Phi(x) = (\rho^L \circ \mathcal{A}^L \circ \rho^{L-1} \circ \cdots \circ \rho^1 \circ \mathcal{A}^1)(x) \tag{2.1}$$

for all $x \in \mathbb{R}^{n_0}$ is called a feedforward artificial neural network of depth L. The network is said to have depth L, width $\max_i n_i$ and activation functions $\rho^1, \ldots, \rho^L$. Furthermore, $W^1, \ldots, W^L$ and $b^1, \ldots, b^L$ are called the weights and biases of the network.

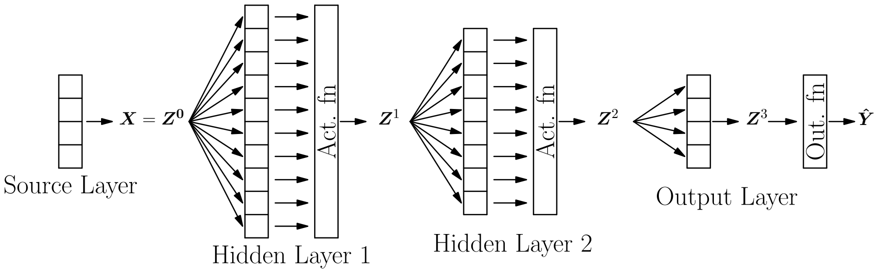


Figure 2.1: An MLP with 2 hidden layers. The source layer transmits the signal X to the first hidden layer. The final output of the network is $\hat{Y}$.

In this terminology, a feedforward ANN of depth L consists of an input layer, $L-1$ hidden layers and an output layer. It is said to be deep if $L \geqslant 3$, in this case we speak of a feedforward deep neural network (DNN). We denote the vector fed into the input layer by Z^0. The l-th layer (with n_l neurons) receives an input vector $Z^{l-1} \in \mathbb{R}^{n_{l-1}}$ and transforms it into the vector $Z^l \in \mathbb{R}^{n_l}$ by first applying the affine linear transformation, followed by the activation function ρ^l,

$$Z^l = \rho^l(W^l Z^{l-1} + b^l), \quad 1 \leqslant l \leqslant L, \tag{2.2}$$

with Z^l serving as the input for the $(l+1)$-th layer. Note that it is only interesting to choose nonlinear activation functions; if all activation functions are linear, the network reduces to an affine linear map. Many suitable options remain, common choices being (the multidimensional version of) the standard logistic function, the hyperbolic tangent and the rectified linear unit (ReLU). This last function is defined as

$$\mathrm{ReLU} : \mathbb{R} \to \mathbb{R} : x \mapsto \max\{x, 0\} = (x)_+, \tag{2.3}$$

and has grown over the past year to the activation function of choice in many applications. Thanks to its simple form and its even simpler derivative, it is computationally cheaper to optimize a network with ReLU activation functions than a network with for instance sigmoidal activation functions. ReLU neural networks also partly resolve the *saturation problem* that is typical for the sigmoidal function: whereas the derivative of a sigmoidal function vanishes for both large positive and large negative values, the derivative of the rectified linear unit only vanishes for negative values. This facilitates again the problem of finding the optimal network, as most optimization algorithms fundamentally depend on the gradient to find an optimum. Furthermore, ReLU neural networks are also able to exactly represent some very basic, but useful functions. This is the topic of Section 2.2. This also allows the construction of explicit approximations of continuous functions, as will be seen in Chapter 3. It is the combination of these theoretical properties and its computational efficiency that makes the ReLU neural network ubiquitous nowadays. As we will almost exclusively use this type of multilayer perceptron in this thesis, we formalize the concept of an ANN that uses the rectified linear unit as activation function.

Definition 2.2 *Let $L, n_0, n_1, \ldots, n_L \in \mathbb{N}$ with $L \geqslant 2$. For $l = 1, 2 \ldots, L-1$, let $\rho^l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ be a function that satisfies $\rho^l(x) = (\mathrm{ReLU}(x_1), \ldots, \mathrm{ReLU}(x_{n_l}))^T$ for all $x \in \mathbb{R}^{n_l}$ and let $\rho^L : \mathbb{R}^{n_L} \to \mathbb{R}^{n_L}$ satisfy $\rho^L(x) = x$ for all $x \in \mathbb{R}^{n_L}$. A feedforward* *artificial neural network of depth L with activation functions $\rho^1, \dots, \rho^L$ is called a ReLU neural network of depth L.*

Definition 2.3 *Let $L, n_0, n_L \in \mathbb{N}$ with $L \geqslant 2$. Then we denote by $\mathfrak{N}_{L,n_0,n_L}$ the set of ReLU neural networks that have depth L, input dimension n_0 and output dimension n_L.*

We can now also make the connection with the problem setting in the introduction of this chapter. The function class under consideration in this case, is of course those of feedforward artificial neural networks. For consistency, we set $n_0 = m$ and $n_L = n$. The parameter space Θ then consists of the weights and biases of the network. Depending on the nature of the problem, the output of the ANN may have to pass through an output function $\mathcal{S}$ to convert the signal into a meaningful form. In classification problems, a suitable choice for such an output function would be the *softmax* function

$$\mathcal{S}(x) : \mathbb{R}^n \to \mathbb{R}^n : x \mapsto \left(\frac{e^{x_1}}{\sum_{j=1}^n e^{x_j}}, \dots, \frac{e^{x_n}}{\sum_{j=1}^n e^{x_j}} \right). \tag{2.4}$$

This choice ensures that the final output vector $\hat{Y} = \mathcal{S}(Z^L)$ satisfies $\sum_{j=1}^n \hat{Y}_j = 1$ and $0 \leqslant \hat{Y}_j \leqslant 1$ for all $1 \leqslant j \leqslant n$, which allows $\hat{Y}_j$ to be viewed as the probability that the input Z^0 belongs to the j-th class. Note that the class predicted by the network is $\arg\max_j \hat{Y}_j$. More output functions, and more information on multilayer perceptrons in general, can be found in Chapter 6 of [Goodfellow et al., 2016].

2.2 Basic ReLU calculus

In the following chapters, we will use ReLU neural networks to construct approximations of certain mappings. For this reason, we must develop some basic results on the composition and addition of neural networks. One can also establish bounds on other properties of these composed networks, e.g. the network connectivitiy (the total number of nonzero entries of weights and biases). As these bounds will play no role in this thesis, we refer the reader to [Grohs et al., 2019a, Grohs et al., 2019b] for the exact results. The following lemma discusses the composition of two ReLU neural networks.

Lemma 2.4 *Let $L_1, L_2, d_1, d_2, N_{L_1}, N_{L_2} \in \mathbb{N}$ with $L_1, L_2 \geqslant 2$, $\Phi_1 \in \mathfrak{N}_{L_1,d_1,N_{L_1}}$, and $\Phi_2 \in \mathfrak{N}_{L_2,d_2,N_{L_2}}$ with $N_{L_1} = d_2$. Then there exists a network $\Psi \in \mathfrak{N}_{L_1+L_2-1,d_1,N_{L_2}}$ satisfying $\Psi(x) = \Phi_2(\Phi_1(x))$, for all $x \in \mathbb{R}^{d_1}$.*

Proof Let $W_i^1, \dots W_i^{L_i}$ and $b_i^1, \dots b_i^{L_i}$ be the weights and biases of Φ_i for $i = 1, 2$ according to Definition 2.1. Define

$$\begin{aligned} &W^j = W_1^j \quad \text{and} \quad b^j = b_1^j \qquad \text{for } j = 1, \dots, L_1 - 1, \\ &W^{L_1} = W_2^1 W_1^{L_1} \quad \text{and} \quad b^{L_1} = W_2^1 b_1^{L_1} + b_2^1, \\ &W^{L_1+j-1} = W_2^j \quad \text{and} \quad b^{L_1+j-1} = b_2^j \qquad \text{for } j = 2, \dots, L_2, \end{aligned} \tag{2.5}$$

and let Ψ be the ReLU neural network corresponding to these weights and biases. □

In many results where the total number of weights and biases plays an important role, Lemma 2.4 should be replaced by Lemma II.5 in [Grohs et al., 2019b]. If the number of neurons in the hidden layers of Φ_1 and Φ_2 are large compared to d_2, the network connectivity of the network of the proof of Lemma 2.4 can be significantly reduced at the cost of creating an additional hidden layer.

As preparation for the result on linear combinations of ReLU neural networks, we show how the depth of a ReLU neural network can be increased in such a way that it still corresponds to the same function.

Lemma 2.5 *Let $L, K, d \in \mathbb{N}$ with $L \geqslant 2$, $\Phi_1 \in \mathfrak{N}_{L,d,1}$, and $K > L$. Then, there exists a corresponding network $\Phi_2 \in \mathfrak{N}_{K,d,1}$ such that $\Phi_2(x) = \Phi_1(x)$, for all $x \in \mathbb{R}^d$.*

Proof Lemma II.6 in [Grohs et al., 2019b]. Let $W_1^1, \dots W_1^L$ and $b_1^1, \dots b_1^L$ be the weights and biases of Φ_1, according to Definition XX. Define

$$\begin{aligned}
&W_2^j = W_1^j \quad \text{and} \quad b_2^j = b_1^j \qquad \text{for } j = 1, \dots, L-1, \\
&W_2^L = \begin{pmatrix} W_1^L \\ -W_1^L \end{pmatrix} \quad \text{and} \quad b_2^L = \begin{pmatrix} b_1^L \\ -b_1^L \end{pmatrix}, \\
&W_2^{L+j} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad b_2^{L+j} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \text{for } j = 1, \dots, K-L-1, \\
&W_2^K = \begin{pmatrix} 1 & -1 \end{pmatrix} \quad \text{and} \quad b_2^K = 0
\end{aligned} \tag{2.6}$$

and let Φ_2 be the ReLU neural network corresponding to these weights and biases. □

Lemma 2.6 *Let $N, L_i, d_i \in \mathbb{R}$, with $L_i \geqslant 2$, $a_i \in \mathbb{R}$, $\Phi_i \in \mathfrak{N}_{L_i,d_i,1}$ for $i = 1, 2, \dots, N$, and set $d = \sum_{i=1}^N d_i$. Then, there exist networks $\Psi_1 \in \mathfrak{N}_{L,d,N}$ and $\Psi_2 \in \mathfrak{N}_{L,d,1}$ with $L = \max_i L_i$, satisfying*

$$\begin{aligned}
\Psi_1(x) &= (a_1\Phi_1(x_1) \quad \dots \quad a_N\Phi_N(x_N))^T \quad \text{and} \\
\Psi_2(x) &= \sum_{i=1}^N a_i \Psi_i(x_i),
\end{aligned} \tag{2.7}$$

for all $x = (x_1^T, \dots, x_N^T)^T \in \mathbb{R}^d$ with $x_i \in \mathbb{R}^{d_i}$ for $i = 1, 2, \dots, N$.

Proof We follow the proof of Lemma II.7 in [Grohs et al., 2019b]. Apply Lemma 2.5 to the networks Φ_i to get corresponding networks $\tilde{\Phi}_i$ of depth L and set $\Psi_1(x) = (a_1\tilde{\Phi}_1(x_1) \quad \dots \quad a_N\tilde{\Phi}_N(x_N))^T$ and $\Psi_2(x) = (1, \dots, 1)\Psi_1(x)$. □

To conclude this chapter, we illustrate how ReLU neural networks can be used to exactly represent some simple functions. The three lemmas above guarantee that any function that can be written as a composition of additions and the rectified linear unit, can in fact be written as a ReLU neural network.

Example 2.7 *The identity function can be written as a ReLU neural network of depth 2 with zero biases and weights $W^1 = (1, -1)^T$ and $W^2 = (1, -1)$. Indeed, the equality $x = (x)_+ - (-x)_+$ holds for all $x \in \mathbb{R}$. Similarly, the absolute value function can be represented by a ReLU neural network of depth 2 with zero biases and weights $W^1 = (1, -1)^T$ and $W^2 = (1, 1)$.*

Example 2.8 *The function $\max : \mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto \max\{x, y\}$ can also be written as a ReLU neural network. One can note that $\max\{x, y\} = \mathrm{id}_{\mathbb{R}}(x) + (y - x)_+$ for $x, y \in \mathbb{R}$.*

Using Example 2.7 and Lemma 2.6, one finds that a possible choice is the ReLU neural network of depth 2 with weights

$$W^1 = \begin{pmatrix} -1 & 1 \\ 1 & 0 \\ -1 & 0 \end{pmatrix} \qquad \textit{and} \qquad W^2 = (1, 1, -1)$$

and zero bias vectors. Moreover, using this result, Lemma 2.4 and Lemma 2.6, we see that also the calculation the maximum of four real numbers can be performed by a ReLU neural network, given the observation that $\max\{a, b, c, d\} = \max\{\max\{a, b\}, \max\{c, d\}\}$ *for* $a, b, c, d \in \mathbb{R}$*.*

Chapter 3

Function approximation with neural networks

Neural networks are nowadays a very popular tool for the approximation of functions, often despite the lack of a rigorous theoretical framework that explains why the results are as good as they are. Nonetheless, it has been the topic of many lines of modern research to answer the following question: how well can a function be approximated by a (ReLU) neural network of a certain architecture? First, we give an overview of some results on this topic for depth-bounded and width-bounded neural networks. Next, we discuss one particular result on the approximation of Lipschitz continuous functions by deep ReLU neural networks [Yarotsky, 2018a].

3.1 Expressive power of neural networks

In the last decades, many papers have been published that show that (deep) neural networks are universal approximators for a multitude of function classes, under varying conditions on the network architecture and the choice of the activation function. Whereas earlier results focus on neural networks of only one hidden layer, more recent results investigate the expressive power of deep neural networks. One of the first famous results is due to Cybenko, who proved in [Cybenko, 1989] that shallow neural networks are universal approximators for continuous functions, under some conditions on the activation function.

Definition 3.1 *A function $\sigma : \mathbb{R} \to \mathbb{R}$ is called discriminatory if for any finite, signed regular Borel measure on $[0,1]^n$ it holds that the condition*

$$\forall y \in \mathbb{R}^n, \theta \in \mathbb{R} : \int_{[0,1]^n} \sigma(y^T x + \theta) d\mu(x) = 0 \tag{3.1}$$

implies that $\mu = 0$.

Theorem 3.2 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous, discriminatory function and let $\alpha_j, \theta_j \in \mathbb{R}, y_j \in \mathbb{R}^n$ for $1 \leqslant j \leqslant n$. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j), \quad x \in \mathbb{R}^n \tag{3.2}$$

are dense in $C([0,1]^n)$.

Cybenko further proves that any continuous, sigmoidal function is discriminatory. It is also not very hard to prove that the ReLU function is discriminatory.

Theoretical results that are in a similar vein as Cybenko's universal approximation theorem were proved in [Funahashi, 1989, Hornik, 1991, Barron, 1993]. Many of these results turned out to be special cases of the following general result:

Theorem 3.3 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a function that is locally essentially bounded with the property that the closure of the set of points of discontinuity has zero Lebesgue measure. For $1 \leqslant j \leqslant n$, let $\alpha_j, \theta_j \in \mathbb{R}$ and $y_j \in \mathbb{R}^n$. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j), \quad x \in \mathbb{R}^n \tag{3.3}$$

are dense in $C(\mathbb{R}^n)$ if and only if σ is not an algebraic polynomial.

This result is due to Leshno, Lin, Pinkus and Schocken [Leshno et al., 1993], the proof relies again on techniques from functional analysis. We conclude that all initial results on the expressive power of neural networks focus on networks with one single hidden layer and merely prove the existence of a network that approximates a given function well. In the last decade, however, the interest in and applications of (deep) neural networks have grown explosively. This sudden massive gain in popularity of a mathematical tool whose theoretical properties are poorly understood, has revived the need of rigorous results on the expressive power of neural networks. As the ReLU activation function became the method of choice in many applications, most recent results only focus on ReLU neural networks. In particular, they provide insight in how the width and depth of the network influence its accuracy. In contrast to the approach of Cybenko and others, these more recent universal approximation theorems are often proved by explicitly constructing the network architecture and weights.

Cybenko already showed that neural networks with bounded depth are expressive enough to approximate continuous functions arbitrarily well. It is then only a natural question to ask whether the same holds for networks with bounded width. A first interesting result on this topic is due to Hu, Lu, Pu, Wang and Wang. In their paper *'The Expressive Power of Neural Networks: A View from the Width'* [Lu et al., 2017], they state the following universal approximation theorem for width-bounded ReLU networks.

Theorem 3.4 *For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and $\varepsilon > 0$, there exists a fully-connected ReLU network f_N with width at most $n+4$, such that*

$$\int_{\mathbb{R}^n} |(f(x) - f_N(x)| dx < \varepsilon. \tag{3.4}$$

The constructed network is a concatenation of subnetworks (so-called blocks), where each block mimics the indicator function on a small enough n-dimensional cube. The result then follows by multiplying the output of each block with a suitable weight. The number of blocks that are needed is indeed finite since the underlying function is Lebesgue-integrable.

A second universal approximation theorem for width-bounded ReLU networks, this time with respect to the supremum norm, can be found in the paper *'Approximating continuous functions by ReLU nets of minimal width'* by to Hanin and Sellke [Hanin and Sellke, 2017]. In addition, they provide bounds on the minimal width needed for the universal approximation property.

Definition 3.5 *Let $m, n \geqslant 1$. The minimal width $w_{min}(n, m)$ is defined as the minimal value of w such that for every continuous function $f : [0, 1]^n \to \mathbb{R}^m$ and every $\varepsilon > 0$ there is a ReLU network f_N with input dimension n, hidden layer widths at most w and output dimension m, such that*

$$\sup_{x \in [0,1]^n} \|f(x) - f_N(x)\| \leqslant \varepsilon. \tag{3.5}$$

The main result of the paper is the following theorem.

Theorem 3.6 *For every $n, m \geqslant 1$,*

$$n + 1 \leqslant w_{min}(n, m) \leqslant n + m. \tag{3.6}$$

In the proof of the upper bound the authors also provide a bound on the depth of the network needed to approximate a continuous function. For this bound we must introduce

$$\omega_f^{-1}(\varepsilon) = \sup\{\delta : \omega_f(\delta) \leqslant \varepsilon\} \tag{3.7}$$

where ω_f is the modulus of continuity of f. Then for any compact set $K \subseteq \mathbb{R}^n$ and continuous function $f : K \to \mathbb{R}^m$, there exists a ReLU network with input dimension n, hidden layer widths $n + m$, output dimension m and depth $O(\mathrm{diam}(K)/\omega_f^{-1}(\varepsilon))^{n+1}$.

Thus far, we have established that both width-bounded and depth-bounded ReLU neural networks admit a universal approximation theorem. It is a natural question to ask which type provides a better accuracy with the same number of neurons. In practice, deeper networks tend to be more powerful than shallow ones. Rolnick and Tegmark provide some interesting results on this topic in the paper *'The power of deeper networks for expressing natural functions'* [Rolnick and Tegmark, 2017].

Definition 3.7 *Suppose that a nonlinear function σ is given. For p a multivariate polynomial, let $M_k(p)$ be the minimum number of neurons in a depth-k artificial neural network f_N that satisfies $\sup_{x \in K} \|p(x) - f_N(x)\| < \varepsilon$ for all $\varepsilon > 0$ on some compact set $K \subset \mathbb{R}^n$.*

Theorem 3.4 of [Rolnick and Tegmark, 2017] proves that $M_k(p)$ is indeed a well-defined finite number. The following theorem then shows that the uniform approximation of monomials requires exponentially more neurons in a shallow than a deep network.

Theorem 3.8 *Let $p(x) = \prod_{i=1}^n x_i^{r_i}$ and set $d = \sum_{i=1}^n r_i$. Suppose that the nonlinearity σ has nonzero Taylor coefficients up to degree $2d$. Then, we have*

1. $M_1(p) = \prod_{i=1}^n (r_i + 1)$,
2. $\min_k M_k(p) \leqslant \sum_{i=1}^n (7\lceil \log_2(r_i) \rceil + 4)$.

Theorem 4.3 of [Rolnick and Tegmark, 2017] provides a generalization of the above theorem to sums of monomials.

Finally, we mention the many recent results of Yarotsky on the approximation of functions by deep neural networks [Yarotsky, 2017, Yarotsky, 2018a, Yarotsky, 2018b, Yarotsky and Zhevnerchuk, 2019]. In [Yarotsky, 2017], he described explicit ReLU neural networks to approximate the function $f : \mathbb{R} \to \mathbb{R} : x \mapsto x^2$ with respect to the supremum norm. This approximation can then be used to approximate the multiplication operator, since $xy = ((x+y)^2 - x^2 - y^2)/2$, and consequently to approximate polynomials and other smooth functions. In [Yarotsky and Zhevnerchuk, 2019], Yarotsky and Zhevnerchuk were able to construct deep neural networks with ReLU and sine activation functions that can approximate Lipschitz functions to exponential accuracy. When restricted to ReLU activation functions, he proved that it is still possible to obtain second-order accuracy using deep neural networks [Yarotsky, 2018a]. Because of its relevance for the subject of this thesis, we will study this result in detail in the next section.

3.2 Approximation of Lipschitz continuous functions by deep ReLU networks

In this section we interpret and summarize the paper *'Optimal approximations of continuous functions by very deep ReLU networks'* by Dmitry Yarotsky [Yarotsky, 2018a] in the special case of Lipschitz continuous functions on the unit interval. Yarotsky constructed for every feasible approximation rate deep ReLU neural networks that can approximate continuous functions on finite-dimensional cubes. He found that the phase diagram of those rates consists of two distinct phases. For slow approximations, a network with constant depth where the weights are continuously assigned is sufficient. This continuity and the constant depth need necessarily to be sacrificed in order to improve the approximation rate.

The main goal of the paper is investigating the relation between the approximation errors, the modulus of continuity of a function and the number of weights W of the approximating network. For a continuous function $f : [0,1]^\nu \to \mathbb{R}$, we define $\tilde{f}_W : [0,1]^\nu \to \mathbb{R}$ to be a ReLU network with W weights and biases that approximates f, where the architecture of the network only depends on W and not on f. The main question becomes then: for which powers $p \in \mathbb{R}$ is it possible to find for all $f \in C([0,1]^\nu)$ a network $\tilde{f}_W$ such that

$$\|f - \tilde{f}_W\|_\infty = O(\omega_f(O(W^{-p}))), \tag{3.8}$$

where ω_f is the modulus of continuity of f? This question is partially answered by the following theorem (Theorem 1 in [Yarotsky, 2018a]). The theorem states that some approximation rates are infeasible, whereas for some other rates discontinuous weight selection is inevitable.

Theorem 3.9 *Approximation rate (3.8) cannot be achieved with $p > \frac{2}{\nu}$. Approximation rate (3.8) can also not be achieved with $p > \frac{1}{\nu}$ if the weights of the approximating network $\tilde{f}$ are required to depend on f continuously with respect to the standard topology of $C([0,1]^\nu)$.*

3.2.1 Approximation with continuous weight assignment

In this section we present the counterpart to the second part of Theorem 3.9, where it was mentioned that when continuous weight selection is required, approximation rate (3.8) is infeasible for $p > \frac{1}{\nu}$. The following theorem states that it is possible to find a ReLU network satisfying the continuity requirement for $p = \frac{1}{\nu}$. We will only discuss the proof in the case of a Lipschitz continuous function on the unit interval.

Theorem 3.10 *There exist network architectures with W weights and, for each W, a weight assignment linear in f such that equation (3.8) is satisfied with $p = \frac{1}{\nu}$. The network architectures can be chosen as consisting of $O(W)$ parallel blocks each having the same architecture that only depends on ν. In particular, the depths of the networks depend on ν but not on W.*

Corollary 3.11 *There exist network architectures with W weights and, for each W, a weight assignment linear in f such that*

$$\|f - \tilde{f}_W\|_\infty \leqslant cL_f W^{-1}, \tag{3.9}$$

where f is a Lipschitz function with Lipschitz constant L_f and $c > 0$ is a constant not depending on f or W. The network architectures can be chosen as consisting of $O(W)$ parallel blocks each having the same architecture. The architecture and depth in particular are independent of all parameters.

Proof Let f be a Lipschitz function on $[0,1]$ with Lipschitz constant L_f. We will construct the piecewise linear approximation of f on some, for now undetermined, scale $\frac{1}{N}$ and prove that the neural network representing this function satisfies the properties of the corollary. First define the intervals $I_n^N = [\frac{n}{N}, \frac{n+1}{N}]$ for $n \in \mathbb{Z}$.[1] Next we define the spike function

$$\varphi : \mathbb{R} \to \mathbb{R} : x \mapsto (\min(1-x, 1+x))_+, \tag{3.10}$$

which is linear on all intervals I_n^1 and satisfies $\varphi(n) = \delta_0(n)$ for all $n \in \mathbb{Z}$. We then define the linear interpolation $\tilde{f}_1$ by

$$\tilde{f}_1(x) = \sum_{n \in \{0,1,\dots N\}} f\left(\frac{n}{N}\right) \varphi\left(Nx - n\right). \tag{3.11}$$

Note that the derivative of $\tilde{f}_1$ in the interior of each interval I_n^N is constant and equal to the difference quotient of f on that interval. From this follows the bound $\|\tilde{f}_1'\|_\infty \leqslant L_f$ and thus $L_{\tilde{f}_1} \leqslant L_f$. Next we consider the discrepancy

$$f_2 = f - \tilde{f}_1, \tag{3.12}$$

for which $L_{f_2} \leqslant 2L_f$. Since every $x \in [0,1]$ is within distance $\frac{1}{2N}$ of one of the interpolation knots where f_2 vanishes, we have that

$$\|f_2\|_\infty \leqslant \frac{1}{2N} L_{f_2} \leqslant \frac{1}{N} L_f. \tag{3.13}$$

[1] In the notation of [Yarotsky, 2018a], $I_n^N = \Delta_{n,\mathrm{id}}^{(N)}$.

Now notice that the spike function can be exactly represented by a small ReLU neural network, a *block*. The desired neural network consists of $N+1$ of such blocks in parallel computing the values $\varphi\left(Nx-n\right)$ for $n \in \{0, 1, \ldots N\}$ and an output layer that returns a weighted sum of those values, with $f(\frac{n}{N})$ as the weights. Clearly, the output of this ReLU network is equal to (3.11). Moreover, the number of weights W is seen to be proportional to the number of grid points N, that is $W = cN$ for some constant $c > 0$. This yields the wanted result. □

3.2.2 Approximation with discontinuous weight assignment

Theorem 3.9 stated that approximation rate (3.8) cannot be achieved for $p > \frac{2}{\nu}$. In this section we will prove that it is possible to find a (deep) ReLU neural network leading to the optimal rate $p = \frac{2}{\nu}$.

Theorem 3.12 *For any $p \in \left(\frac{1}{\nu}, \frac{2}{\nu}\right]$ there exist architectures with depths $L = O(W^{p\nu-1})$ and respective weight assignments such that inequality (3.8) holds with this p. In particular, the rate with $p = \frac{2}{\nu}$ can be achieved by narrow fully-connected architectures of constant width $2\nu + 10$.*

We will again concentrate on the approximation rate result for Lipschitz functions on the unit interval. We refer the reader to [Yarotsky, 2018a] for the proof of the general case and a proof of the statements concerning the precise width and depth of an example of a suitable network.

Corollary 3.13 *There exist narrow fully-connected architectures such that for every Lipschitz function on the unit interval with Lipschitz constant L_f and for every number of weights W it holds that*

$$\|f - \tilde{f}_W\|_\infty \leqslant cL_f W^{-2}, \tag{3.14}$$

where $c > 0$ is an independent constant.

Proof We split the proof into three parts. In the first part we propose an approximation method. Next we guarantee that this approximation can indeed be represented by a ReLU neural network and in the final part we verify that the number of weights and the approximation error satisfy the bound of the corollary.

Step 1: The two-scales approximation and its accuracy

Let f be a Lipschitz function on the unit interval with Lipschitz constant L_f. We will approximate f in two significantly different steps, which will both require not more than $W/2$ weights. The first step merely consists of approximating f by $\tilde{f}_1$ (3.11) with $N \leqslant c_1 W$, where c_1 is a sufficiently small constant. We now proceed by decomposing the discrepancy $f_2 = f - \tilde{f}_1$ and approximating the components of this decomposition. A key insight will be that it suffices to dispose only over a discrete set of approximating functions.

Let $S = \{0, 1, 2\}$ and define for every $q \in S$ the function given by

$$g_q(x) = \sum_{n \in (q+3\mathbb{Z}) \cap [0,N]} \varphi(Nx - n), \tag{3.15}$$

for $x \in [0,1]$ and where φ is the spike function defined in (3.10). It can easily be noted that $\sum_{q\in S} g_q = 1$ on the unit interval. We then decompose the discrepancy as

$$f_2 = \sum_{q\in S} f_{2,q} \quad \text{where} \quad f_{2,q} = f_2 g_q. \tag{3.16}$$

Let $Q_n = \left[\frac{n-1}{N}, \frac{n+1}{N}\right]$. The support of $f_{2,q}$ is then given by the disjoint union

$$Q^q = \bigcup_{n\in(q+3\mathbb{Z})\cap[0,N]} Q_n. \tag{3.17}$$

It will also turn out to be useful to define $q(n)$ as the unique $q \in S$ such that $n \in (q+3\mathbb{Z}) \cap [0,N]$. We now calculate the Lipschitz constant of $f_{2,q}$. Let $x, y \in [0,1]$. Then

$$\begin{aligned} |f_{2,q}(x) - f_{2,q}(y)| &\leqslant \|f_2\|_\infty |g_q(x) - g_q(y)| + \|g_q\|_\infty |f_2(x) - f_2(y)| \\ &\leqslant \frac{L_f}{N} N|x-y| + L_{f_2}|x-y| \\ &\leqslant 3L_f|x-y|, \end{aligned} \tag{3.18}$$

where we used in particular (3.13), $L_{g_q} = N$ and $L_{f_2} \leqslant 2L_f$. We conclude that $L_{f_{2,q}} \leqslant 3L_f$.

We now proceed by approximating each $f_{2,q}$ by $\tilde{f}_{2,q}$ on the grid $\mathbb{Z}/M$, with $\frac{M}{N} \in \mathbb{Z}$, such that it is a refinement of the grid $\mathbb{Z}/N$. We define $\tilde{f}_{2,q}$ on the grid $\mathbb{Z}/M$ by rounding down $f_{2,q}$ to the nearest multiple of the discretisation parameter $\lambda = \frac{3L_f}{M}$,

$$\tilde{f}_{2,q}\left(\frac{m}{M}\right) = \lambda \lfloor f_{2,q}\left(\frac{m}{M}\right)/\lambda \rfloor, \quad m \in [0,1,\ldots,M]. \tag{3.19}$$

We then define $\tilde{f}_{2,q}$ on $[0,1]$ as the piecewise linear interpolation of these values. We also define $\hat{f}_{2,q}$ as the piecewise linear interpolation of $f_{2,q}$ on the grid $\mathbb{Z}/M$. Note that this directly yields the bound $\|\hat{f}_{2,q} - \tilde{f}_{2,q}\|_\infty \leqslant \lambda$. We also have

$$\|\hat{f}_{2,q} - f_{2,q}\|_\infty \leqslant \frac{1}{2M} L_{\hat{f}_{2,q} - f_{2,q}} \leqslant \frac{1}{M} L_{f_{2,q}} \leqslant \lambda, \tag{3.20}$$

where we used similar arguments as in the proof of Theorem 1. Altogether we have

$$\|f - \tilde{f}\|_\infty \leqslant \sum_{q\in S} \|f_{2,q} - \tilde{f}_{2,q}\|_\infty \leqslant 6\lambda = \frac{18L_f}{M}. \tag{3.21}$$

If we now succeed in taking $W \sim \sqrt{M}$ we obtain the wanted approximation rate.

Step 2: Representation of the refined approximation

Recall that on the refined grid, $\tilde{f}_{2,q}$ was defined as a multiple of the parameter λ. We will store and characterize this value by this integer multiplier of λ. For $q \in S$ and $n \in (q+3\mathbb{Z}) \cap [0,N]$ (and therefore for the interval Q_n) we write

$$A_{q,n}(m) = \left\lfloor f_{2,q}\left(\frac{n}{N} + \frac{m}{M}\right)/\lambda \right\rfloor, \quad m \in \left[-\frac{M}{N}, \ldots, \frac{M}{N}\right]. \tag{3.22}$$

Note that $A_{q,n}(\pm\frac{M}{N})=0$. Furthermore we define

$$B_{q,n}(m)=A_{q,n}(m)-A_{q,n}(m-1), \quad m\in\left[-\frac{M}{N}+1,\dots,\frac{M}{N}-1\right]. \tag{3.23}$$

It is now crucial to recall that $L_{f_{2,q}}\leqslant M\lambda$. This allows us to see that $B_{q,n}(m)\in\{-1,0,1\}$ and therefore store all these coefficients in a single ternary number

$$b_{q,n}=\sum_{t=1}^{2M/N-1}3^{-t}\left(B_{q,n}\left(t-\frac{M}{N}\right)+1\right). \tag{3.24}$$

This representation also allows to recover $\{B_{q,n}(m)\}$ from $b_{q,n}$ using ReLU neural networks. Define the sequence z_t recursively by setting $z_0=b_{q,n}$ and $z_{t+1}=3z_t-\lfloor 3z_t\rfloor$, in this way $B_{q,n}\left(t-\frac{M}{N}\right)=\lfloor 3z_{t-1}\rfloor$. Since there exists $\varepsilon>0$ such that $0\leqslant z_t<1-\varepsilon$, we do not need to be able to write the floor function as a ReLU neural network on the whole interval $[0,3]$. In fact, we can replace it by a piecewise linear function that coincides with the floor function for all possible values of $3z_t$. Details can be found in [Yarotsky, 2018a].

We now proceed by rewriting $\tilde{f}_{2,q}$ on the interval Q_n. For this reason we define

$$\Phi_n(m,x)=\sum_{s=-M/N+1}^{m}\varphi\left(M\left(x-\left(\frac{n}{N}+\frac{s}{M}\right)\right)\right). \tag{3.25}$$

Summation by parts allows us to write

$$\begin{aligned}\tilde{f}_{2,q}(x)&=\lambda\sum_{m=-M/N+1}^{M/N-1}\varphi\left(M\left(x-\left(\frac{n}{N}+\frac{m}{M}\right)\right)\right)A_{q,n}(m)\\&=\lambda\sum_{m=-M/N+1}^{M/N-1}\left(\Phi_n(m,x)-\Phi_n(m-1,x)\right)A_{q,n}(m)\\&=\lambda\sum_{m=-M/N+1}^{M/N-1}\Phi_n(m,x)B_{q,n}(m)\end{aligned} \tag{3.26}$$

for $x\in Q_n$. We now wish to make this representation independent of the chosen interval Q_n. As a first step we can define

$$b_q(x)=\sum_{n\in(q+3\mathbb{Z})\cap[0,N]}b_{q,n}((2-|Nx-n|)_+-(1-|Nx-n|)_+). \tag{3.27}$$

The second factor of the summand is 1 if $x\in Q_n$ and 0 on $Q^q\setminus Q_n$. Using a similar method (i.e. by using a function that acts like a step function on Q^q) one can define a function Ψ_q that maps all $x\in Q^q$ to the index of the interval it belongs to. That is, if $x\in Q_n$ then $\Psi_{q(n)}(x)=n$. Consequently we get $b_q(x)=b_{q,\Psi_q(x)}$.

Similarly, we want to replace $\Phi_n(m,x)$ in (3.26) by an expression independent of n. We will call this expression $\tilde{\Phi}_q(m,x)$. In particular it should satisfy for every n that

$\tilde{\Phi}_{q(n)}(m,x) = \Phi_n(m,x)$ on $Q^{q(n)}$ and $\tilde{\Phi}_{q(n)}(m,x) = 0$ on $(Q^{q(n)})^c$. This can be achieved by defining

$$\tilde{\Phi}_q(m,x) = \min(\Phi_{\Psi_q(x)}(m,x), \theta_q(x)) \tag{3.28}$$

where θ_q is a piecewise linear function that vanishes on $(Q^q)^c$ and is larger than the first argument of the minimum on Q^q. Therefore we can write

$$\tilde{f}_{2,q}(x) = \lambda \sum_{m=-M/N+1}^{M/N-1} \tilde{\Phi}_q(m,x) B_q(m,x). \tag{3.29}$$

The last obstacle is the computation of the product inside the sum, but this can be overcome by noting that $\tilde{\Phi}_q(m,x) \in [0,1]$, $B_q(m,x) \in \{-1,0,1\}$ and that for all $x \in [0,1]$ and $y \in \{-1,0,1\}$ we can write

$$xy = (x+y-1)_+ + (-x-y)_+ - (-y)_+. \tag{3.30}$$

We can write our total approximation as

$$\tilde{f} = \tilde{f}_1 + \sum_{q \in S} \tilde{f}_{2,q} \tag{3.31}$$

and we have proven that it can be exactly computed by a ReLU neural network.

Step 3: Network implementation

Recall that we need to show that the total number of weights used to implement $\tilde{f}_2$ does not exceed $W/2$. We will count for all defined quantities in the above construction how many weights are needed.

We need to compute $O(1)$ times the term $\tilde{f}_{2,q}$. In order to compute $\theta_q(x)$, $\Psi_q(x)$ and $b_q(x)$ we need to sum $O(N)$ expressions that each require $O(1)$ network weights. Let $c_2 > 0$ be the constant such that if we take $N \leqslant c_2 W$, then this part of the network consists of less than $W/4$ weights. As we also need $N \leqslant c_1 W$ in order to control the number of weights needed to compute $\tilde{f}_1$, we fix $N = \min(c_1, c_2)W$.

Recovering $B_{q,n}(m)$ from $b_q(x)$ for every $m \in \left[-\frac{M}{N}+1, \dots, \frac{M}{N}-1\right]$ requires $\frac{2M}{N}-1$ times $O(1)$ network weights. Similarly, calculating all $\Phi_n(m,x)$ requires also $O(M/N)$ weights. The same holds for computing the sum to obtain $\tilde{f}_{2,q}(x)$. Take now $c_3 > 0$ and $M = c_3 W^2$ such that this part of the network needs at most $W/4$. We have now described a network with at most W weights such that for some $c > 0$, $\|f - \tilde{f}_W\|_\infty \leqslant c L_f W^{-2}$, where we used (3.21).

It is also interesting to note that all sums can be computed in a serial way. Therefore the described network is very deep, but it has a narrow width that is independent of the function. In fact, following the arguments in [Yarotsky, 2018a], one can prove that the width of the network does not need to exceed 12. □

Chapter 4

ENO interpolation with ReLU neural networks

The previous chapter has clearly demonstrated the expressive power of ReLU neural networks, in particular their capability of approximating rough functions. In practice, there might be a major issue in this approach when used to approximate an unknown function $f : D \subseteq \mathbb{R} \to \mathbb{R}$ based on a finite set $\mathbb{S} \subset \{(x, f(x)) : x \in D\}$, as for each individual function f a new neural network has to be found. The retrieval of this network, also called the *training* of the network, has a computational cost that is significantly higher than that of other classical regression methods, which makes this approach rather impractical. This motivates us to investigate how one can obtain a neural network that takes input in D and produces an output interpolant, or rather its evaluation at certain sample points. Such a network primarily depends on the training data $\mathbb{S}$ and can be reused for each individual function. Instead of creating an entirely novel data dependent interpolation procedure, we base ourselves in this chapter on the essentially non-oscillatory (ENO) procedure of [Harten et al., 1987]. This procedure can attain any order of accuracy for smooth functions and reduces to first-order accuracy for continuous rough functions. In what follows, we introduce the ENO interpolation framework and we suggest new approaches that use ReLU neural networks to reproduce or approximate the results of the ENO interpolation procedure.

4.1 ENO interpolation

We begin by introducing the ENO interpolation procedure and its properties. Let f be a function on $\Omega = [c, d] \subset \mathbb{R}$ that is at least p times continuously differentiable. We define a sequence of nested uniform grids $\{\mathcal{T}^k\}_{k=0}^{K}$ on Ω, where

$$\mathcal{T}^k = \{x_i^k\}_{i=0}^{N_k}, \quad I_i^k = [x_{i-1}^k, x_i^k], \quad x_i^k = c + ih_k, \quad h_k = \frac{(d-c)}{N_k}, \quad N_k = 2^k N_0, \quad (4.1)$$

for $0 \leqslant i \leqslant N_k$, $0 \leqslant k \leqslant K$ and some positive integer N_0. Furthermore we define $f^k = (f(x_0^k), \ldots, f(x_{N_k}^k))$, $f_i^k = f(x_i^k)$ and we let $f_{-p+2}^k, ..., f_{-1}^k$ and $f_{N_k+1}^k, ...f_{N_k+p-2}^k$ be suitably prescribed ghost values. We are interested in finding an interpolation operator $\mathcal{I}^{h_k}$ such that

$$\mathcal{I}^{h_k} f(x) = f(x) \text{ for } x \in \mathcal{T}^k \quad \text{and} \quad \|\mathcal{I}^{h_k} f - f\|_\infty = O(h_k^p) \text{ for } k \to \infty.$$

In standard approximation theory, this is achieved by defining $\mathcal{I}^{h_k}f$ on I_i^k as the unique polynomial p_i^k of degree $p-1$ that agrees with f on a chosen set of p points, including x_{i-1}^k and x_i^k. The linear interpolant ($p=2$) can be uniquely obtained using the stencil $\{x_{i-1}^k, x_i^k\}$. However, there are several candidate stencils to choose from when $p>2$. The ENO interpolation procedure considers the stencil sets

$$\mathcal{S}_i^p(r) = \{x_{i-1-r+j}^k\}_{j=0}^{p-1}, \quad 0 \leqslant r \leqslant p-2, \tag{4.2}$$

where r is called the (left) stencil shift. Note that every stencil shift r uniquely defines a polynomial $\mathcal{P}[\mathcal{S}_i^p(r)]$ of degree $p-1$ that perfectly agrees with f on $\mathcal{S}_i^p(r)$. To get insight in the smoothness of this polynomial, we consider divided differences $f[z_0,\dots,z_m]$ and undivided differences $\Delta_f[z_0,\dots,z_m]$. These are inductively defined by

$$\begin{aligned} f[z] &= \Delta_f[z] = f(z), \\ \Delta_f[z_0,\dots,z_m] &= \Delta_f[z_1,\dots,z_m] - \Delta_f[z_0,\dots,z_{m-1}], \\ f[z_0,\dots,z_m] &= \frac{\Delta_f[z_0,\dots,z_m]}{z_m - z_0}, \end{aligned} \tag{4.3}$$

for $z, z_0, \dots, z_m \in [c,d]$ and $m \in \mathbb{N}$. One can calculate that for any permutation $\sigma : \{0,\dots,m\} \to \{0,\dots,m\}$ it holds $f[z_{\sigma(0)},\dots,z_{\sigma(m)}] = f[z_0,\dots,z_m]$. This allows us to define

$$f[\mathcal{S}_i^p(r)] = f[x_{i-1-r}^k,\dots,x_{i-r+p-2}^k]. \tag{4.4}$$

We can introduce a similar definition for undivided differences,

$$\Delta_f[\mathcal{S}_i^p(r)] = \Delta_f[x_{i-1-r}^k,\dots,x_{i-r+p-2}^k]. \tag{4.5}$$

where the order of the arguments of $\Delta_f[\cdot]$ are not to be interchanged, as undivided differences are not invariant under permutations. Under the assumption that $z_i = z_0 + i\Delta m$ for $1 \leqslant i \leqslant m$, the divided differences reflect the smoothness of f in the following way.

1. If f is at least m times continuously differentiable on $[z_0, z_m]$, then there exists $\xi \in [z_0, z_m]$ with
$$f[z_0,\dots,z_m] = \frac{f^{(m)}(\xi)}{m!}. \tag{4.6}$$

2. If f is only piecewise smooth and (for $0 \leqslant j \leqslant m-1$) the j-th derivative of f has one discontinuity at $z \in [z_0, z_m]$, then [Laney, 2007]
$$f[z_0,\dots,z_m] = \mathcal{O}\left(\frac{1}{(\Delta z)^{m-j}}\left(\frac{d^j f(z+)}{dx^j} - \frac{d^j f(z-)}{dx^j}\right)\right). \tag{4.7}$$

Using this, we can write the polynomial $\mathcal{P}[\mathcal{S}_i^p(r)]$ as Newton's divided differences interpolation polynomial,

$$\mathcal{P}[\mathcal{S}_i^p(r)](x) = \sum_{m=0}^{p-1} f[\mathcal{S}_i^{m+1}(r)] \prod_{j=0}^{m-1} (x - x_{i-1-r+j}^k), \tag{4.8}$$

and we observe that [Harten, 1986]

$$\begin{aligned}\mathcal{P}[\mathcal{S}_i^p(r)](x) &= \mathcal{P}[\mathcal{S}_i^{p-1}(r)](x) + f\,[\mathcal{S}_i^p(r)]\prod_{j=0}^{p-2}(x - x_{i-1-r+j}^k) \quad \text{and} \\ \mathcal{P}[\mathcal{S}_i^p(r+1)](x) &= \mathcal{P}[\mathcal{S}_i^{p-1}(r)](x) + f[\mathcal{S}_i^p(r+1)]\prod_{j=0}^{p-2}(x - x_{i-1-r+j}^k).\end{aligned} \tag{4.9}$$

This allows us to inductively define the p-th order accurate ENO interpolation polynomial. For $p = 2$, only $\mathcal{S}_i^2(0)$ is a suitable stencil set. Therefore we define the second-order ENO interpolation polynomial on I_i^k by $\mathcal{P}[\mathcal{S}_i^2(0)]$. For $p = 3$, one can choose between $\mathcal{S}_i^3(0)$ and $\mathcal{S}_i^3(1)$. From (4.9), it follows that the only difference between $\mathcal{P}[\mathcal{S}_i^3(0)]$ and $\mathcal{P}[\mathcal{S}_i^3(1)]$ is the coefficient of the quadratic term. Since this coefficient mirrors the smoothness of f, as established earlier, we take $\mathcal{P}[\mathcal{S}_i^3(0)]$ as third-order ENO polynomial if $|f[\mathcal{S}_i^3(0)]| \leqslant |f[\mathcal{S}_i^3(1)]|$ and $\mathcal{P}[\mathcal{S}_i^3(1)]$ otherwise. We thus extended the stencil in the direction of smoothness. By repeating this process, one obtains a stencil shift r_i^k that uniquely defines the p-th-order accurate ENO interpolation polynomial $p_i^k = \mathcal{P}[\mathcal{S}_i^p(r_i^k)]$. Selecting an ENO polynomial thus reduces to $p-2$ comparisons of two (un)divided differences, as described by Algorithm 1. Note that the use of undivided instead of divided differences has no effect on the output of Algorithm 1. We can write the final interpolant as

$$\mathcal{I}^{h_k} f(x) = \sum_{i=1}^{N_k} p_i^k(x) \mathbb{1}_{[x_{i-1}^k, x_i^k)}(x).$$

It is clear from its construction, that the ENO polynomial is designed to be as smooth as possible. Whereas many classical interpolation methods will lead to spurious oscillations when applied on rough functions (e.g. near discontinuities), this will not be the case for the *essentially non-oscillatory* (ENO) polynomial. This property can be made precise using the total variation (TV) seminorm

$$TV_c^d(f) = \sup\left\{\sum_{i=0}^{N-1}(f(x_{i+1}) - f(x_i)) : c = x_0 < \ldots < x_N = d,\ N \in \mathbb{N}\right\} \tag{4.10}$$

of a function $f : [c,d] \to \mathbb{R}$. We say that an interpolation operator $\mathcal{I}^h$ is *total variation bounded* (TVB) when there exists a function $g : [c,d] \to \mathbb{R}$ such that

$$\|\mathcal{I}^h f - g\|_\infty = O(h) \text{ for } h \to 0 \quad \text{and} \quad TV_c^d(g) \leqslant TV_c^d(f). \tag{4.11}$$

It is this property that guarantees the disappearance of spurious oscillations when the grid is refined. In addition, $\mathcal{I}^h f$ is monotone on intervals containing a discontinuity [Shu, 1998]. Both are very useful properties to have when interpolating a rough function.

Algorithm 1 ENO interpolation stencil selection

```
Input: ENO order $p$, input array $\Delta^0 = \{f^k_{i+j}\}_{j=-p+1}^{p-2}$, for any $0 \leqslant i \leqslant N_k$.
Output: Stencil shift $r$.
  Evaluate Newton undivided differences:
  for $j = 1$ to $p-1$ do
    $\Delta^j = \Delta^{j-1}[2 : \text{end}] - \Delta^{j-1}[1 : \text{end} - 1]$
  end for
  Find shift:
  $r = 0$
  for $j = 2$ to $p-1$ do
    if $|\Delta^j[p-2-r]| < |\Delta^j[p-1-r]|$ then
      $r = r+1$
    end if
  end for
  return $r$
```

Remark 4.1 *It is also interesting to make the connection between selecting an ENO polynomial and the following optimization problem:*

$$\begin{aligned} &\text{minimize} \quad \mathcal{F}(r_2, \dots, r_p) := \sum_{j=2}^{p} |f[\mathcal{S}_i^j(r_j)]| \\ &\textit{such that} \quad 0 \leqslant r_j \leqslant j-2 \quad \text{for} \quad 2 \leqslant j \leqslant p, \\ &\qquad\qquad \mathcal{S}_i^{j-1}(r_{j-1}) \subseteq \mathcal{S}_i^j(r_j) \quad \text{for} \quad 3 \leqslant j \leqslant p. \end{aligned} \tag{4.12}$$

Algorithm 1 then can be interpreted as a greedy algorithm with the intent to find the solution of (4.12)*.*

In many applications, one is only interested in predicting the values of f^{k+1} given f^k. In this case, there is no need to calculate $\mathcal{I}^{h_{k+1}} f$ and evaluate it on $\mathcal{T}^{k+1}$. Instead, one can use Lagrangian interpolation theory to see that there exist fixed coefficients $C^p_{r,j}$ such that

$$\begin{aligned} \mathcal{I}^{h_k} f(x^{k+1}_{2i-1}) &= \sum_{j=0}^{p-1} C^p_{r^k_i, j} f^k_{i-r^k_i+j} \quad \text{for} \ \ 1 \leqslant i \leqslant N_k \quad \text{and} \\ \mathcal{I}^{h_k} f(x^{k+1}_{2i}) &= f^{k+1}_{2i} = f^k_i \ \ \text{for} \ \ 0 \leqslant i \leqslant N_k, \end{aligned} \tag{4.13}$$

where r^k_i is the stencil shift corresponding to the smoothest stencil for interval I^k_i. The coefficients $C^p_{r,j}$ are listed in Table 4.1.

4.2 ENO reconstruction

ENO was actually initially introduced by [Harten et al., 1987] for high-order accurate piecewise polynomial reconstruction given cell averages of a function. This allows to develop high-order accurate numerical methods for hyperbolic conservation laws, the so-called ENO schemes. ENO reconstruction can be loosely interpreted as ENO interpolation applied to the primitive function. For the sake of completeness, we repeat the main steps of the algorithm for reconstruction purposes.

p	r	$j=0$	$j=1$	$j=2$	$j=3$
3	0	3/8	3/4	-1/8	-
	1	-1/8	3/4	3/8	-
4	0	5/16	15/16	-5/16	1/16
	1	-1/16	9/16	9/16	-1/16
	2	1/16	-5/16	15/16	5/16

Table 4.1: Coefficients for ENO interpolation for $p > 2$ used in (4.13).

Let V be a function on $[c, d]$. We define a uniform mesh $\mathcal{T}$ on $[c, d]$ with N cells,

$$\mathcal{T} = \{I_i\}_{i=1}^N, \quad I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}], \quad x_i = c + (2i-1)\frac{h}{2}, \quad h = \frac{(d-c)}{N}, \tag{4.14}$$

for $1 \leqslant i \leqslant N$ and where x_i and $x_{i\pm\frac{1}{2}}$ denote the cell center and cell interfaces of the cell I_i, respectively. We are given the cell averages

$$\overline{V}_i = \frac{1}{h}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} V(\xi)\mathrm{d}\xi, \quad 1 \leqslant i \leqslant N \tag{4.15}$$

and we define $\overline{V}_{-p+2}, ..., \overline{V}_0$ and $\overline{V}_{N+1}, ..., \overline{V}_{N+p-1}$ to be suitably prescribed ghost values. The goal is to find a local interpolation operator $\mathcal{I}_i^h$ such that

$$\|\mathcal{I}_i^h V - V\|_{\infty, I_i} = O(h^p) \text{ for } h \to 0.$$

For this purpose, let $\hat{V}$ be the primitive function of V and note that we have access to the value of $\hat{V}$ at the cell interfaces,

$$\hat{V}(x_{i+\frac{1}{2}}) = h\sum_{j=0}^{i} \overline{V}_j \quad \text{where} \quad \hat{V}(x) = \int_c^x V(\xi)\mathrm{d}\xi. \tag{4.16}$$

Next let P_i be the unique polynomial of degree p that agrees with $\hat{V}$ on a chosen set of $p+1$ cell interfaces that includes $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. The ENO reconstruction procedure considers the stencil sets

$$\mathcal{S}_i^r = \{x_{i-\frac{1}{2}-r+j}\}_{j=0}^p, \quad 0 \leqslant r \leqslant p-1,$$

where r is called the (left) stencil shift. The smoothest stencil is then selected based on the local smoothness of f using Newton's undivided differences. Algorithm 2 describes how the stencil shift r_i corresponding to this stencil can be obtained. Note that r_i uniquely defines the polynomial P_i. We then define $\mathcal{I}_i^h V$ to be the first derivative of P_i, one can check that this polynomial is indeed a p-th order accurate approximation. Note that the interpolants on two adjacent intervals do not need to agree on the mutual cell interface.

In order to implement an ENO scheme, one only needs the values of $\mathcal{I}_i^h V$ at the cell interfaces $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. Analogously to (4.13), these can be directly obtained by calculating

$$\mathcal{I}_i^h V(x_{i+\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i,j}^p \overline{V}_{i-r_i+j} \quad \text{and} \quad \mathcal{I}_i^h V(x_{i-\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i-1,j}^p \overline{V}_{i-r_i+j}, \tag{4.17}$$

where r_i is stencil shift corresponding to the smoothest stencil for interval I_i and with the coefficients $\tilde{C}_{r,j}^p$ listed in Table 4.2.

Algorithm 2 ENO reconstruction stencil selection

```
Input: ENO order $p$, input array $\Delta^0 = \{\overline{V}_{i+j}\}_{j=-p+1}^{p-1}$, for any $1 \leqslant i \leqslant N$.
Output: Stencil shift $r$.
  Evaluate Newton undivided differences:
  for $j = 1$ to $p-1$ do
    $\Delta^j = \Delta^{j-1}[2 : \text{end}] - \Delta^{j-1}[1 : \text{end} - 1]$
  end for
  Find shift:
  $r = 0$
  for $j = 1$ to $p-1$ do
    if $|\Delta^j[p-1-r]| < |\Delta^j[p-r]|$ then
      $r = r + 1$
    end if
  end for
  return $r$
```

p	r	$j=0$	$j=1$	$j=2$	$j=3$
2	-1	3/2	-1/2	-	-
	0	1/2	1/2	-	-
	1	-1/2	3/2	-	-
3	-1	11/6	-7/6	1/3	-
	0	1/3	5/6	-1/6	-
	1	-1/6	5/6	1/3	-
	2	1/3	-7/6	11/6	-
4	-1	25/12	-23/12	13/12	-1/4
	0	-1/4	13/12	-5/12	1/12
	1	-1/12	7/12	7/12	-1/12
	2	1/12	-5/12	13/12	-1/4
	3	-1/4	13/12	-23/12	25/12

Table 4.2: Coefficients for ENO reconstruction used in (4.17).

4.3 ENO stencil selection with ReLU neural networks

The crucial step of the ENO procedure is determining the correct stencil shift. Given the stencil shift, the retrieval of the ENO polynomial is straightforward. ENO-p can therefore be interpreted as a classification problem, with the goal of mapping an input vector (the evaluation of a certain function on a number of points) to one of the $p-1$ classes (the stencil shifts). One of the main results of this thesis is that the ENO stencil shift can exactly be calculated using a ReLU neural network. The result is joint work with Deep Ray and can also be found in [De Ryck et al., 2020].

Theorem 4.2 *There exists a ReLU neural network consisting of* $p + \left\lceil \log_2 \binom{p-2}{\lfloor \frac{p-2}{2} \rfloor} \right\rceil$ *hidden layers and a suitable output function, that takes input* $\Delta^0 = \{f_{i+j}^k\}_{j=-p+1}^{p-2}$ *and leads to exactly the same stencil shift as the one obtained by Algorithm 1.*

Proof We look for the ENO stencil shift $r := r_i^k$ corresponding to the interval I_i^k. Let $k \in \mathbb{N}$ and define $\Delta_j^0 = f_{i+j}^k$ for $-p+1 \leqslant j \leqslant p-2$ and $0 \leqslant i \leqslant N_k$, where

$f^k_{-p+1}, \dots, f^k_{-1}$ and $f^k_{N_k+1}, \dots, f^k_{N_k+p-2}$ are suitably defined ghost values. Furthermore we define $\Delta^s_j = \Delta^{s-1}_j - \Delta^{s-1}_{j-1}$ for s odd and $\Delta^s_j = \Delta^{s-1}_{j+1} - \Delta^{s-1}_j$ for s even. In what follows, we use Y^l and Z^l to denote the values of the l-th layer of the neural network before and after activation, respectively. We use the notation X^l for an auxiliary vector needed to calculate Y^l.

Step 1. Take the input to the network to be

$$Z^0 = [\Delta^0_{-p+1}, \dots, \Delta^0_{p-2}] \in \mathbb{R}^{2(p-1)}.$$

These are all the candidate function values considered in Algorithm 1.

Step 2. We want to obtain all quantities Δ^s_j that are compared in Algorithm 1, as shown in Figure 4.1. We therefore choose the first layer (before activation) to be

$$Y^1 = \begin{pmatrix} Y_\Delta \\ -Y_\Delta \end{pmatrix} \in \mathbb{R}^{2M} \quad \text{where} \quad Y_\Delta = \begin{pmatrix} \Delta^2_0 \\ \Delta^2_{-1} \\ \vdots \end{pmatrix} \in \mathbb{R}^M$$

is the vector of all the terms compared in Algorithm 1 and $M = \frac{p(p-1)}{2} - 1$. Note that every undivided difference is a linear combination of the network input. Therefore one can obtain Y^1 from Z^0 by taking a null bias vector and weight matrix $W^1 \in \mathbb{R}^{2M \times (2p-2)}$. After applying the ReLU activation function, we obtain

$$Z^1 = \begin{pmatrix} (Y_\Delta)_+ \\ (-Y_\Delta)_+ \end{pmatrix}.$$

Step 3. We next construct a vector $X^2 \in \mathbb{R}^L$, where $L = \frac{(p-2)(p-1)}{2}$, that contains all the quantities of the if-statement in Algorithm 1. This is ensured by setting,

$$X^2 = \begin{pmatrix} |\Delta^2_{-1}| - |\Delta^2_0| \\ |\Delta^3_0| - |\Delta^3_1| \\ |\Delta^3_{-1}| - |\Delta^3_0| \\ \vdots \end{pmatrix}.$$

Keeping in mind that $|a| = (a)_+ + (-a)_+$ for $a \in \mathbb{R}$ we see that there is a matrix $\widetilde{W}^2 \in \mathbb{R}^{L \times 2M}$ such that $X^2 = \widetilde{W}^2 Z^1$. We wish to quantify for each component of X^2 whether it is strictly negative or not (cf. the if-statement of Algorithm 1). For this reason, we define the functions $H_1 : \mathbb{R} \to \mathbb{R}$ and $H_2 : \mathbb{R} \to \mathbb{R}$ by

$$H_1(x) = \begin{cases} 0 & x \leqslant -1 \\ x+1 & -1 < x < 0 \\ 1 & x \geqslant 0 \end{cases} \quad \text{and} \quad H_2(x) = \begin{cases} -1 & x \leqslant 0 \\ x-1 & 0 < x < 1 \\ 0 & x \geqslant 1 \end{cases}.$$

The key property of these functions is that H_1 and H_2 agree with the Heaviside function on $x > 0$ and $x < 0$, respectively. When $x = 0$ the output is respectively $+1$ and -1. Now note that $H_1(x) = (x+1)_+ - (x)_+$ and $H_2(x) = (x)_+ - (x-1)_+ - 1$. This motivates us to define

$$Y^2 = \begin{pmatrix} X^2 + 1 \\ X^2 \\ X^2 - 1 \end{pmatrix} \in \mathbb{R}^{3L},$$

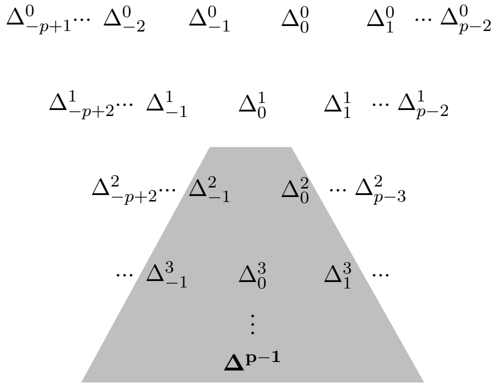

$\Delta^0_{-p+1} \cdots \Delta^0_{-2}$
Δ^0_{-1}
Δ^0_0
$\Delta^0_1 \cdots \Delta^0_{p-2}$
$\Delta^1_{-p+2} \cdots \Delta^1_{-1}$
Δ^1_0
$\Delta^1_1 \cdots \Delta^1_{p-2}$
$\Delta^2_{-p+2} \cdots \Delta^2_{-1}$
$\Delta^2_0 \cdots \Delta^2_{p-3}$
$\cdots \Delta^3_{-1}$
Δ^3_0
$\Delta^3_1 \cdots$
$\vdots$
$\mathbf{\Delta^{p-1}}$

Figure 4.1: Only undivided differences in the shaded region are compared in Algorithm 1.

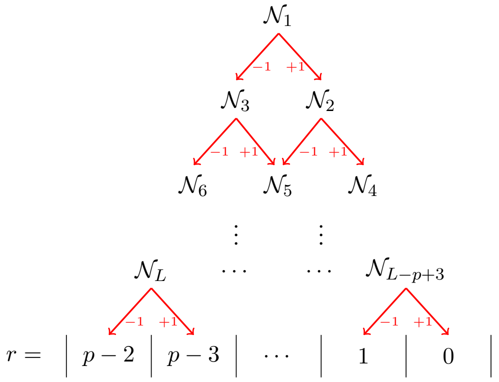


Figure 4.2: Arrangement of $\mathcal{N}_1, \dots, \mathcal{N}_L$ into directed acyclic graph.

which can be obtained from Z^1 by taking weight matrix $W^2 \in \mathbb{R}^{3L \times 2K}$ and bias vector $b^2 \in \mathbb{R}^{3L}$,

$$W^2 = \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \otimes \mathbb{I}_L \right) \cdot \widetilde{W}^2 \quad \text{and} \quad b^2_j = \begin{cases} 1 & 1 \leqslant j \leqslant L \\ 0 & L+1 \leqslant j \leqslant 2L \\ -1 & 2L+1 \leqslant j \leqslant 3L \end{cases}$$

where $\mathbb{I}_L$ denotes the $L \times L$ unit matrix. After activation we obtain $Z^2 = (Y^2)_+ = (W^2 Z^1 + b^2)_+$.

Step 4. We first define $X^3 \in \mathbb{R}^{2L}$ by

$$X^3_j = \begin{cases} H_1(X^2_j) = Z^2_j - Z^2_{L+j} & 1 \leqslant j \leqslant L \\ H_2(X^2_{j-L}) = Z^2_j - Z^2_{L+j} - 1 & L+1 \leqslant j \leqslant 2L. \end{cases}$$

This is clearly for every j an affine transformation of the entries of Z^2. For this reason there exist a matrix $\widetilde{W}^3 \in \mathbb{R}^{2L \times 3L}$ and a bias vector $\tilde{b}^3 \in \mathbb{R}^{2L}$ such that $X^3 = \widetilde{W}^3 Z^2 + \tilde{b}^3$. In order to visualize the next steps, we arrange the elements of X^3 in a triangular directed acyclic graph, shown in Figure 4.2, where every node $\mathcal{N}_j$ corresponds to the tuple $(X^3_j, X^3_{j+L}) = (H_1(X^2_j), H_2(X^2_j))$. We note that this tuple is either of the form $(+1, H_2(X^2_j))$ or $(H_1(X^2_j), -1)$. Algorithm 1 is equivalent to finding a path from the top node to one of the bins on the bottom. Starting from $\mathcal{N}_1$, we move to the closest element to the right in the row below (i.e. $\mathcal{N}_2$) if $\mathcal{N}_1$ is of the form $(+1, H_2(X^2_j))$. If $\mathcal{N}_1$ is of the form $(H_1(X^2_j), -1)$, we move to the closest element to the left in the row below (i.e. $\mathcal{N}_3$). If $\mathcal{N}_1$ is of the form $(+1, -1)$, then it is not important in which direction we move. Both paths lead to a suitable ENO stencil shift. Repeating the same procedure at each row, one ends up in one of the $p-1$ bins at the bottom representing the stencil shift r.

There are 2^{p-2} paths from the top to one of the bins at the bottom. In order to represent the path using a $(p-2)$-tuple of entries of X^3, one needs to choose between $H_1(X^2_j)$ and $H_2(X^2_j)$ at every node of the path, leading to 2^{p-2} variants of each path. At least one of these variants only takes the values $+1$ and -1 on the nodes and is identical to the path described above; this is the variant we wish to select. Counting all variants, the total number of paths is 2^{2p-4}.

Consider a path $\mathcal{P} = (X^3_{j_1}, \ldots, X^3_{j_{p-2}})$ that leads to bin r. We define for this path a weight vector $W \in \{-1, 0, 1\}^{2L}$ whose elements are set as

$$W_j = \begin{cases} +1 & \text{if } X^3_j = +1 \text{ and } j = j_s \text{ for some } 1 \leqslant s \leqslant p-2 \\ -1 & \text{if } X^3_j = -1 \text{ and } j = j_s \text{ for some } 1 \leqslant s \leqslant p-2 \\ 0 & \text{otherwise.} \end{cases}$$

For this particular weight vector and for any possible $X^3 \in \mathbb{R}^{2L}$ we have $W \cdot X^3 \leqslant p-2$, with equality achieved if and only if the entries of X^3 appearing in $\mathcal{P}$ are assigned the precise values used to construct W. One can construct such a weight vector for each of the 2^{2p-4} paths. We next construct the weight matrix $\widehat{W}^3 \in \mathbb{R}^{2^{2p-4} \times 2L}$ in such a way that the first $2^{p-2} \cdot \binom{p-2}{0}$ rows correspond to the weight vectors for paths reaching $r = 0$, the next $2^{p-2} \cdot \binom{p-2}{1}$ for paths reaching $r = 1$ et cetera. We also construct the bias vector $\hat{b}^3 \in \mathbb{R}^{2^{2p-4}}$ by setting each element to $p-2$ and we define $\hat{X}^3 = \widehat{W}^3 X^3 + \hat{b}^3 = \widehat{W}^3(\widetilde{W}^3 Z^2 + \tilde{b}^3) + \hat{b}^3$. By construction, $\hat{X}^3_j = 2p-4$ if and only if path j corresponds to a suitable ENO stencil shift, otherwise $0 \leqslant \hat{X}^3_j < 2p-4$.

Step 5. Next the components of $\hat{Z}^3$ that correspond to the same bin need to be accumulated. We will use the following lemma.

Lemma 4.3 *Let* $\max_n : \mathbb{R}^n_+ \to \mathbb{R}$ *be the maximum of* n *positive real numbers. Then the function* $\max_n$ *can be written as a ReLU neural network with* $\lceil \log_2 n \rceil$ *hidden layers with widths* $2^{\lceil \log_2 n \rceil}, \ldots, 2^1$.

Proof We prove the statement for the case $n = 2^m$, where $m \in \mathbb{N}$, by induction on m. Let $x \in \mathbb{R}^2_+$. Then $\max(x_1, x_2) = (x_1)_+ + (x_2 - x_1)_+$. This can indeed be written as a ReLU network with 1 hidden layer of width 2.

Now assume the statement holds for $n = 2^m$ and take $x \in \mathbb{R}^{2^{m+1}}_+$. By the induction hypothesis, there exist two ReLU networks like in the statement that compute $z_1 = \max(x_1, \ldots, x_{2^m})$ and $z_2 = \max(x_{2^m+1}, \ldots, x_{2^{m+1}})$. Construct a new ReLU network by putting these two networks in parallel and adding a hidden layer with the values z_1 and $z_2 - z_1$ before activation. The formula $\max(z_1, z_2) = (z_1)_+ + (z_2 - z_1)_+$ then gives the wanted result. □

The definitions of $Y^3, Y^4 \ldots$ follow from the lemma above. We can now define the final output vector in the following way,

$$\hat{Y}_j = \max_{n(j)} \left(\hat{Z}^3 \left(1 + 2^{p-2} \cdot \sum_{k=0}^{j-2} \binom{p-2}{k} \right), \ldots, \hat{Z}^3 \left(2^{p-2} \cdot \sum_{k=0}^{j-1} \binom{p-2}{k} \right) \right),$$

where $n(j) = 2^{p-2} \cdot \binom{p-2}{j-1}$ and $\hat{Z}^3(j) = \hat{Z}^3_j$. Note that the calculation of $\hat{Y}$ requires $\left\lceil \log_2 n \left(\lfloor \frac{p-2}{2} \rfloor + 1 \right) \right\rceil = p-2+ \left\lceil \log_2 \binom{p-2}{\lfloor \frac{p-2}{2} \rfloor} \right\rceil$ additional hidden layers. By construction, it is true that $\hat{Y}_j = 2p-4$ if and only if the $(j-1)$-th bin is reached. Furthermore, $\hat{Y}_j < 2p-4$ if the $(j-1)$-th bin is not reached. The set of all suitable stencil shifts R and the unique stencil shift r from Algorithm 1 are then respectively given by

$$R = \text{argmax}_j \hat{Y}_j - 1 \quad \text{and} \quad r = \min R = \min \text{argmax}_j \hat{Y}_j - 1. \tag{4.18}$$

□

Remark 4.4 *The neural network constructed in the above theorem is local in the sense that for each interval, it provides a stencil shift. These local neural networks can be concatenated to form a single neural network that takes as its input the vector f^k of sampled values and returns the vector of interpolated values that approximates f^{k+1}. The global neural network combines the output stencil shift of each local neural network with a simple linear mapping* (4.13)*.*

Although the previous theorem provides a network architecture for every order p, the obtained networks are excessively large for small p. We therefore present alternative constructions for ENO interpolation of orders $p = 3, 4$. These results are due to Deep Ray.

Algorithm 1 for $p = 3$ can be exactly represented by the following ReLU network with a single hidden layer, whose architecture is given by:

- Input $X = (\Delta^0_{-2}, \Delta^0_{-1}, \Delta^0_0, \Delta^0_1)^\top$.
- The first hidden layer is identical to the one described in the original proof of Theorem 4.2 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{4\times 4}$,

$$W^1 = \begin{pmatrix} 0 & 1 & -2 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{pmatrix}, \quad b^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.19}$$

- Output layer:

$$W^2 = \begin{pmatrix} -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}, \quad b^2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.20}$$

- The shift is determined using (4.18), since

$$\hat{Y} = \begin{pmatrix} |\Delta^2_{-1}| - |\Delta^2_0| \\ |\Delta^2_0| - |\Delta^2_{-1}| \end{pmatrix}.$$

For $p = 4$, Algorithm 1 can be represented by following ReLU network with 3 hidden layers:

- Input $X = (\Delta^0_{-3}, \Delta^0_{-2}, \Delta^0_{-1}, \Delta^0_0, \Delta^0_1, \Delta^0_2)^\top$.
- The first hidden layer is identical to the one described in the original proof of Theorem 4.2 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{10\times 6}$,

$$W^1 = \begin{pmatrix} \widetilde{W}^1 \\ -\widetilde{W}^1 \end{pmatrix} \in \mathbb{R}^{10\times 6} \quad \text{where} \quad \widetilde{W}^1 = \begin{pmatrix} 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -3 & 1 \\ 0 & -1 & 3 & -3 & 1 & 0 \\ -1 & 3 & -3 & 1 & 0 & 0 \end{pmatrix}. \tag{4.21}$$

- The second hidden layer has a null bias and the weight vector

$$W^2 = \begin{pmatrix} \widetilde{W}^2 & \widetilde{W}^2 \\ -\widetilde{W}^2 & -\widetilde{W}^2 \end{pmatrix} \in \mathbb{R}^{6\times 10} \text{ where } \widetilde{W}^2 = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}. \tag{4.22}$$

Note that $\widetilde{W}^2 \in \mathbb{R}^{3\times 5}$ also comes back in the original proof of Theorem 4.2 for $p = 4$.

- The third hidden layer:

$$W^3 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ -1 & 1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad b^3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.23}$$

- Output layer:

$$W^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad b^4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.24}$$

- After an elementary, yet tedious case study, one can show that the shift can again be determined using (4.18).

Remark 4.5 *Similarly, one can show that Algorithm 2 for $p = 2$ can be exactly represented by a ReLU DNN with one hidden layer of width 4. The input and output dimension are 3 and 2, respectively. For $p = 3$, Algorithm 2 can be shown to correspond to a ReLU DNN with three hidden layers of dimensions $(10, 6, 4)$. Input and output dimension are 5 and 4, respectively.*

4.4 ENO interpolation with ReLU neural networks

After having successfully recast the ENO stencil selection as a ReLU neural network, it is natural to investigate whether there exists a ReLU neural network with output $(\mathcal{I}^{h_k} f)^{k+1}$, as in the setting of (4.13) in Section 4.1. Since ENO interpolation is a discontinuous procedure and a ReLU neural network is a continuous function, a network with such an output does not exist. It remains however interesting to investigate to which extent we can approximate ENO using ReLU neural networks.

As a first step in developing such an approximation, we provide an explicit formula to calculate $(\mathcal{I}^{h_k} f)_i^{k+1}$. We base ourselves on the inductive definition of the ENO polynomials, given in (4.9), and the interpretation using a graph from Figure 4.2 from the proof of Theorem 4.2. We also introduce for $\varepsilon \geqslant 0$ the function $H_\varepsilon : \mathbb{R} \to \mathbb{R}$ defined by

$$H_\varepsilon(x) = \begin{cases} 0 & x \leqslant 0 \\ x/\varepsilon & 0 < x \leqslant \varepsilon \\ 1 & x < \varepsilon. \end{cases} \tag{4.25}$$

In Figure 4.3, we revisit the directed acyclic graph displayed in Figure 4.2 from the proof of Theorem 4.2. Note that we altered the notation and turned the arrows around. We are going to define a mapping $\mathcal{Q}^p$ that connects every node in the graph to one of the polynomials of $\{\mathcal{P}[\mathcal{S}_i^p(r)] : 0 \leqslant r \leqslant p-2\}$. We start with the bottom layer of the triangle and we set

$$\mathcal{Q}^p[\mathcal{N}_r^p](x) = \mathcal{P}[\mathcal{S}_i^p(r)](x) \tag{4.26}$$

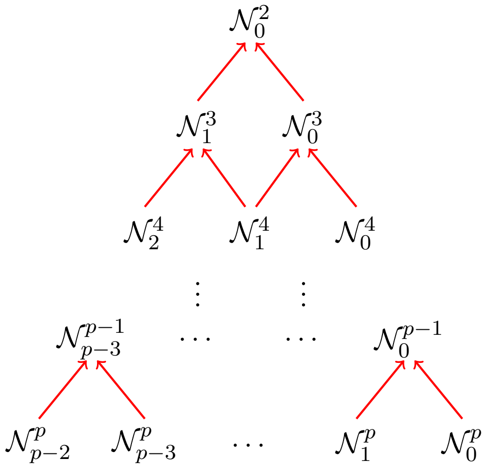


Figure 4.3: Directed acyclic graph that can be used during the calculation of the ENO interpolant.

for $0 \leqslant r \leqslant p-2$. Every node $\mathcal{N}_r^m$ of a higher layer is connected to two nodes from the layer below. Each of these two nodes corresponds through $\mathcal{Q}^p$ to a polynomial; we define $\mathcal{Q}^p[\mathcal{N}_r^m]$ to be the smoothest of these two polynomials. We use undivided differences as smoothness indicator, as explained in Section 4.1. This procedure is repeated until the top node of the graph is reached. The polynomial corresponding to $\mathcal{N}_0^2$ then corresponds to the ENO polynomial. This procedure can be quantified by the formula

$$\begin{aligned}\mathcal{Q}^p[\mathcal{N}_r^m](x) &= (1-\alpha_r^m)\cdot\mathcal{Q}^p[\mathcal{N}_r^{m+1}](x) + \alpha_r^m \cdot \mathcal{Q}^p[\mathcal{N}_{r+1}^{m+1}](x) &\quad (4.27\text{a})\\ &= \mathcal{Q}^p[\mathcal{N}_r^{m+1}](x) + \alpha_r^m \cdot \left(\mathcal{Q}^p[\mathcal{N}_{r+1}^{m+1}](x) - \mathcal{Q}^p[\mathcal{N}_r^{m+1}](x)\right), &\quad (4.27\text{b})\end{aligned}$$

where the coefficients α_m^r are explicitly given by

$$\alpha_r^m = H_0\left(|\Delta_f[\mathcal{S}_i^{m+1}(r)]| - |\Delta_f[\mathcal{S}_i^{m+1}(r+1)]|\right), \tag{4.28}$$

for $2 \leqslant m \leqslant p-1$ and $0 \leqslant r \leqslant m-2$. One can check that indeed

$$\mathcal{Q}^p[\mathcal{N}_0^2](x) = \mathcal{P}[\mathcal{S}_i^p(r_i^k)](x), \tag{4.29}$$

where r_i^k is the ENO stencil shift. Note that this formula is independent of the grid size for uniform grids. For the base case $m = p$ (4.26), we note that $\mathcal{P}[\mathcal{S}_i^p(r)](x^*)$ can (for every r) be trivially written as ReLU neural network, as it merely is a linear combination with fixed coefficients of the function values of the stencil. On the other hand, as we have already established, the formula of the induction step cannot be calculated using a pure ReLU neural network. Nevertheless, we will base ourselves on this formula to introduce an approximate ENO algorithm that can be exactly written as a ReLU neural network. A first straightforward step is to replace H_0 by H_ε with $\varepsilon > 0$, since

$$H_\varepsilon(x) = \frac{1}{\varepsilon}(x)_+ - \frac{1}{\varepsilon}(x-\varepsilon)_+, \tag{4.30}$$

which can be written as a ReLU network. This then leads to the definition

$$\alpha_{r,\varepsilon}^m = H_\varepsilon\left(|f[\mathcal{S}_i^{m+1}(r)]| - |f[\mathcal{S}_i^{m+1}(r+1)]|\right), \tag{4.31}$$

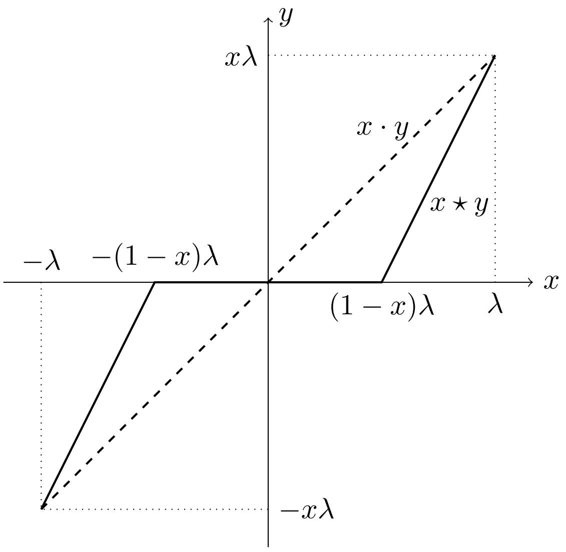


Figure 4.4: Plot of $x \star y$ and $x \cdot y$ for fixed $x \in [0,1]$ and $\lambda > 0$.

for $2 \leqslant m \leqslant p-1$ and $0 \leqslant r \leqslant m-2$. The only remaining issue is that the multiplication of two numbers cannot be exactly represented using a ReLU neural network. We therefore introduce the following operation in order to replace the multiplication of bounded numbers.

Definition 4.6 *For $\lambda > 0$, we denote by $\star$ the operation on $[0,1] \times [-\lambda, \lambda]$ given by*

$$\star : [0,1] \times [-\lambda, \lambda] \to [-\lambda, \lambda] : (x, y) \mapsto x \star y := (y + \lambda x - \lambda)_+ - (-y + \lambda x - \lambda)_+. \quad (4.32)$$

We plot $x \star y$ for fixed $x \in [0,1]$ and $\lambda > 0$ in Figure 4.4. Next, we list some properties of $\star$.

Lemma 4.7 *For $\lambda > 0$, the operation $\star$ satisfies the following properties:*

1. *For all $x \in \{0,1\}$ and $y \in [-\lambda, \lambda]$ it holds true that $x \star y = xy$.*
2. *For all $x \in [0,1]$ and $y \in [0, \lambda]$ we have $0 \leqslant x \star y \leqslant xy$, for all $x \in [0,1]$ and $y \in [-\lambda, 0]$ we have $xy \leqslant x \star y \leqslant 0$.*
3. *There exist $x \in [0,1]$ and $y_1, y_2 \in [-\lambda, \lambda]$ such that*
$$\min\{y_1, y_2\} \leqslant (1-x) \star y_1 + x \star y_2 \leqslant \max\{y_1, y_2\}$$
does not hold.
4. *For all $x \in [0,1]$ and $y_1, y_2 \in [-\lambda, \lambda]$ it holds true that*
$$\min\{y_1, y_2\} \leqslant y_1 + x \star (y_2 - y_1) \leqslant \max\{y_1, y_2\}.$$

Proof Properties 1 and 2 follow immediately from the definition and can also be verified on Figure 4.4. For the third property, note that $1/2 \star \lambda/2 + 1/2 \star \lambda/2 = 0$. Property 4 is an application of property 2. □

Lemma 4.7 allows us to discuss the effect of replacing the multiplication $\cdot$ by the operation $\star$ in the definition of $\mathcal{Q}^p[\mathcal{N}_r^m]$ in (4.27). First of all, we check whether this replacement is well-defined. If we assume that f is a bounded function, then it follows that there exists a constant $\lambda > 0$ such that all ENO approximations at some chosen point lie in the interval $[-\lambda, \lambda]$. From its definition, we also note that $0 \leqslant \alpha_{r,\varepsilon}^m \leqslant 1$. Therefore it is possible to replace $\cdot$ by $\star$. Furthermore, if we do not replace H_0 by H_ε, then the first property of Lemma 4.7 assures that the introduction of $\star$ has no effect on the value of $\mathcal{Q}^p[\mathcal{N}_r^m]$ as $\alpha_r^m \in \{0, 1\}$. The third and fourth property of Lemma 4.7 stress that $\star$ is not distributive over the addition anymore. Therefore replacing $\cdot$ by $\star$ in (4.27a) will not lead to the same quantity as doing so in (4.27b). We will choose to base ourselves on variant (4.27b), as the fourth property of Lemma 4.7 guarantees that

$$\mathcal{Q}_\varepsilon^p[\mathcal{N}_r^m](x) = \mathcal{Q}_\varepsilon^p[\mathcal{N}_r^{m+1}](x) + \alpha_{r,\varepsilon}^m \star \left(\mathcal{Q}_\varepsilon^p[\mathcal{N}_{r+1}^{m+1}](x) - \mathcal{Q}_\varepsilon^p[\mathcal{N}_r^{m+1}](x)\right) \tag{4.33}$$

will still be a convex combination of $\mathcal{Q}_\varepsilon^p[\mathcal{N}_{r+1}^{m+1}](x)$ and $\mathcal{Q}_\varepsilon^p[\mathcal{N}_r^{m+1}](x)$, which is required for the approximation to be meaningful. Together with (4.31), (4.33) and

$$\mathcal{Q}_\varepsilon^p[\mathcal{N}_r^p](x) = \mathcal{P}[\mathcal{S}_i^p(r)](x), \tag{4.34}$$

we can define our approximation of the ENO interpolation polynomial as $\mathcal{Q}_\varepsilon^p[\mathcal{N}_0^2]$. Figure 4.5 visualizes a possible ReLU neural network for the calculation of $\mathcal{Q}_\varepsilon^p[\mathcal{N}_0^2]$ in the case of fourth-order ENO. It consists of four hidden layer with widths 13, 9, 5 and 3. The flowchart shows the values of the neurons of each layer before activation. Similar networks can be constructed for other orders. The network for ENO-3 has for instance 3 hidden layers with widths 6, 4 and 3.

Although our approximation is a very straightforward generalization of the ENO procedure, it can not be guaranteed that $\|\mathcal{Q}_\varepsilon^p[\mathcal{N}_0^2] - \mathcal{Q}_\varepsilon^p[\mathcal{N}_0^2]\|_\infty$ is small. One of the reasons for this is that absolute values of undivided differences are compared in the ENO procedure, and not the undivided differences directly. In the latter case, an error bound would have been possible. Nevertheless, our approximation of ENO does have some desirable properties. First, we recall that our goal is to train a network with the architecture from above on a finite training data set, consisting of function evaluations on a stencil. For every such data set, the parameter $\varepsilon > 0$ can be taken small enough such that every $\alpha_{r,\varepsilon}^m$ is either 0 or 1. As a consequence, the approximation $\mathcal{Q}_\varepsilon^p[\mathcal{N}_0^2]$ agrees with the ENO approximation for every stencil in this finite set, which guarantees that it is theoretically possible to reach an accuracy of 100% on the training data set. This guarantee of course does not hold anymore for new functions. Fortunately, it can be proven that for $\varepsilon > 0$ small enough and a fine enough grid, our approximation of ENO agrees with the original ENO stencil selection procedure in the neighbourhood of jump discontinuities. The following proposition makes this statement precise.

Proposition 4.8 *Let $m \in \mathbb{N}$ with $m \geqslant 2$ and let $f_- : [c,d] \to \mathbb{R}$ and $f_+ : [c,d] \to \mathbb{R}$ be smooth functions. Let $h > 0$ and $x_0 \in [c,d]$ be such that*

$$x_i = x_0 + ih \quad \text{for } 0 \leqslant i \leqslant m+1,$$

all lie in $[c,d]$ and set $\mathcal{S}_i^{m+1}(r+1) = \{x_0, \ldots, x_m\}$ and $\mathcal{S}_i^{m+1}(r) = \{x_1, \ldots, x_{m+1}\}$, using notation as in (4.2). Let also $x_m < z \leqslant x_{m+1}$ and define

$$f(x) = \begin{cases} f_-(x) & x \leqslant z \\ f_+(x) & x > z \end{cases}$$

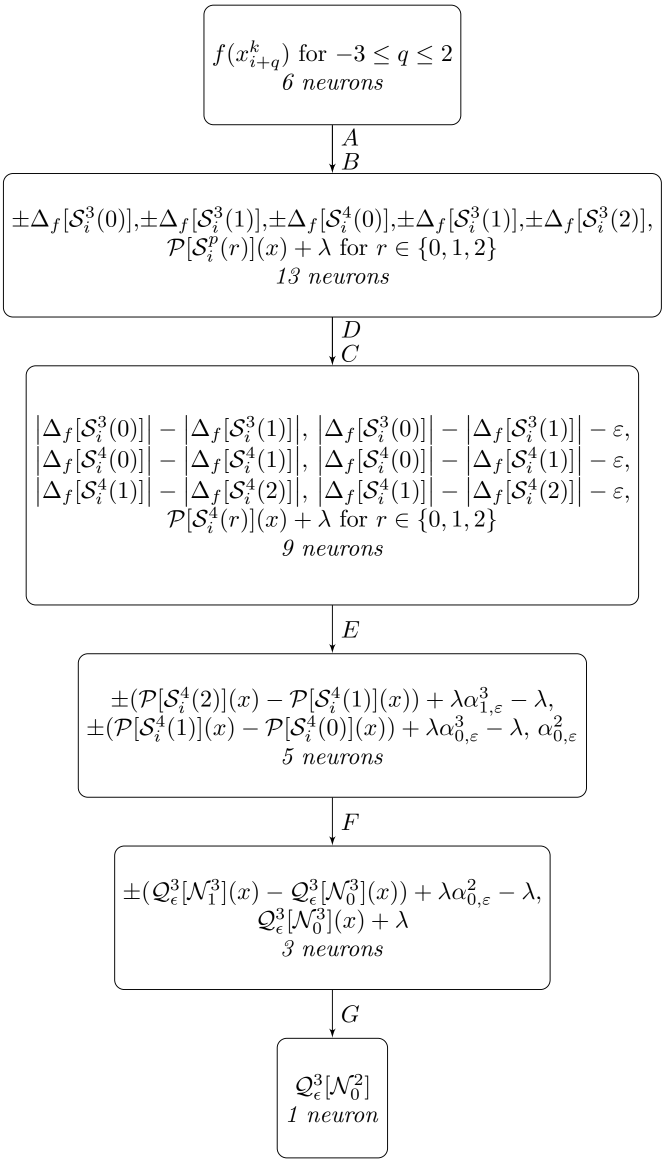

$f(x^k_{i+q})$ for $-3 \leq q \leq 2$
6 neurons
A
B
$\pm\Delta_f[\mathcal{S}^3_i(0)]$,$\pm\Delta_f[\mathcal{S}^3_i(1)]$,$\pm\Delta_f[\mathcal{S}^4_i(0)]$,$\pm\Delta_f[\mathcal{S}^3_i(1)]$,$\pm\Delta_f[\mathcal{S}^3_i(2)]$,
$\mathcal{P}[\mathcal{S}^p_i(r)](x) + \lambda$ for $r \in \{0,1,2\}$
13 neurons
D
C
$\big|\Delta_f[\mathcal{S}^3_i(0)]\big| - \big|\Delta_f[\mathcal{S}^3_i(1)]\big|$, $\big|\Delta_f[\mathcal{S}^3_i(0)]\big| - \big|\Delta_f[\mathcal{S}^3_i(1)]\big| - \varepsilon$,
$\big|\Delta_f[\mathcal{S}^4_i(0)]\big| - \big|\Delta_f[\mathcal{S}^4_i(1)]\big|$, $\big|\Delta_f[\mathcal{S}^4_i(0)]\big| - \big|\Delta_f[\mathcal{S}^4_i(1)]\big| - \varepsilon$,
$\big|\Delta_f[\mathcal{S}^4_i(1)]\big| - \big|\Delta_f[\mathcal{S}^4_i(2)]\big|$, $\big|\Delta_f[\mathcal{S}^4_i(1)]\big| - \big|\Delta_f[\mathcal{S}^4_i(2)]\big| - \varepsilon$,
$\mathcal{P}[\mathcal{S}^4_i(r)](x) + \lambda$ for $r \in \{0,1,2\}$
9 neurons
E
$\pm(\mathcal{P}[\mathcal{S}^4_i(2)](x) - \mathcal{P}[\mathcal{S}^4_i(1)](x)) + \lambda\alpha^3_{1,\varepsilon} - \lambda$,
$\pm(\mathcal{P}[\mathcal{S}^4_i(1)](x) - \mathcal{P}[\mathcal{S}^4_i(0)](x)) + \lambda\alpha^3_{0,\varepsilon} - \lambda$, $\alpha^2_{0,\varepsilon}$
5 neurons
F
$\pm(\mathcal{Q}^3_\epsilon[\mathcal{N}^3_1](x) - \mathcal{Q}^3_\epsilon[\mathcal{N}^3_0](x)) + \lambda\alpha^2_{0,\varepsilon} - \lambda$,
$\mathcal{Q}^3_\epsilon[\mathcal{N}^3_0](x) + \lambda$
3 neurons
G
$\mathcal{Q}^3_\epsilon[\mathcal{N}^2_0]$
1 neuron

Figure 4.5: ReLU neural network to calculate the approximation $\mathcal{Q}^p_\varepsilon[\mathcal{N}^2_0]$ for ENO-4.

for $x \in [c,d]$. Furthermore we set $[f] = f_+(z) - f_-(z)$ and

$$\mu = \sup_{[c,d]} |f'_-| + \sup_{[c,d]} |f'_+| + 2 \sup_{[c,d]} |f_-^{(m)}|. \tag{4.35}$$

For $0 < \varepsilon < |[f]|$, the condition

$$h < \min \left\{ 1, \frac{|[f]| - \varepsilon}{\mu} \right\}$$

implies that

$$|\Delta_f[\mathcal{S}_i^{m+1}(r)]| - |\Delta_f[\mathcal{S}_i^{m+1}(r+1)]| > \varepsilon$$

and therefore that $\alpha^m_{r,\varepsilon} = \alpha^m_r$, where α^m_r and $\alpha^m_{r,\varepsilon}$ are resp. defined in (4.28) *and* (4.31).

Proof We adopt the definitions and notation from the proposition statement and set $\mathcal{S}_1 = \mathcal{S}_i^{m+1}(r+1)$ and $\mathcal{S}_2 = \mathcal{S}_i^{m+1}(r)$ to simplify notation. We then have that,

$$\begin{aligned} f(x_i) &= f_-(x_i) \quad \text{for } 0 \leqslant i \leqslant m, \\ f(x_{m+1}) &= f_+(x_{m+1}) = f_-(x_{m+1}) + [f] + (x_{m+1} - z)(f'_+(\nu_1) - f'_-(\nu_2)) \end{aligned} \tag{4.36}$$

for some $\nu_1, \nu_2 \in [z, x_{m+1}]$. By (4.3) and (4.6) we then have the following identities for the undivided differences,

$$\begin{aligned} \Delta_f[\mathcal{S}_1] &= h^m f_-^{(m)}(\xi_1), \\ \Delta_f[\mathcal{S}_2] &= h^m f_-^{(m)}(\xi_2) + [f] + (x_{m+1} - z)(f'_+(\nu_1) - f'_-(\nu_2)) \end{aligned} \tag{4.37}$$

for some $\xi_1 \in [x_0, x_m]$ and $\xi_2 \in [x_1, x_{m+1}]$. We now look for a condition on h such that $|\Delta_f[\mathcal{S}_2]| - |\Delta_f[\mathcal{S}_1]| > \varepsilon$. Indeed, then we have that $\alpha^m_{r,\varepsilon} = \alpha^m_r$ for the corresponding r. We calculate that under the assumption that $0 < h < 1$ and using (4.35),

$$\begin{aligned} |\Delta_f[\mathcal{S}_2]| - |\Delta_f[\mathcal{S}_1]| &\geqslant |[f]| - |x_{m+1} - z||f'_+(\nu_1) - f'_-(\nu_2)| - |h^m f_-^{(m)}(\xi_1)| - |h^m f_-^{(m)}(\xi_2)| \\ &\geqslant |[f]| - h(|f'_+(\nu_1)| + |f'_-(\nu_2)|) - 2h \sup_{x \in [c,d]} |f_-^{(m)}| \\ &\geqslant |[f]| - h\mu. \end{aligned} \tag{4.38}$$

From this it follows easily that $|\Delta_f[\mathcal{S}_2]| - |\Delta_f[\mathcal{S}_1]| > \varepsilon$ when $h < \min\{1, (|[f]| - \varepsilon)/\mu\}$.□

Chapter 5

Second-order ENO-SR interpolation with ReLU neural networks

In the previous chapter we introduced the ENO interpolation method and highlighted its ability to interpolate rough functions without undesirable side effects (e.g. oscillations near discontinuities). There is however still room for improvement. By itself, the ENO interpolation procedure degrades to first-order accuracy for piecewise smooth functions i.e, functions with a singularity in the second derivative. However, following [Harten, 1989], one can use sub-cell resolution, together with ENO interpolation, to obtain second-order accurate approximation of such functions [Aràndiga et al., 2005]. Similar to Section 4.3, we argue in this chapter that ENO-SR interpolation can be interpreted as a classification problem and prove the existence of a ReLU neural network that predicts the right class. In addition, we propose a variant of the ENO-SR procedure of [Aràndiga et al., 2005] and recast it as a deep neural network, both in classification and regression context. In the following, we assume f to be a continuous function that is two times differentiable except at a single point z where the first derivative has a jump of size $[f'] = f'(z+) - f'(z-)$. We use the notation introduced in Section 4.1.

5.1 Second-order ENO-SR interpolation

As a first step, we investigate the second-order ENO-SR procedure as proposed in [Aràndiga et al., 2005]. We denote the ENO-SR-2 interpolation of some function f by $\mathcal{I}^{h_k} f$. In what follows, we first describe the algorithm and list some properties thereof. Similarly to Section 4.3, we prove that the ENO-SR-2 stencil shift can be exactly obtained using a ReLU neural network.

5.1.1 Algorithm

We first state the ENO-SR detection and interpolation mechanism, as proposed in [Aràndiga et al., 2005]. The mechanism uses second-order differences as smoothness indicator and labels every interval as good (G) or bad (B) after comparing adjacent second-order differences. For a sufficiently fine grid, it can be shown that the interval that contains z is labelled as B (Lemma 2 in [Aràndiga et al., 2005]). The central idea is to construct a piecewise linear interpolant only based on G intervals, which will result in second-order accuracy on all intervals.

We define

$$\Delta_h^2 f(x) = f(x-h) - 2f(x) + f(x+h).$$

The rules of the detection mechanism are the following:

1. If
$$|\Delta_{h_k}^2 f(x_{i-1}^k)| > \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i-1\pm n}^k)|$$
then both I_{i-1}^k and I_i^k are initially labelled B.
2. If
$$|\Delta_{h_k}^2 f(x_i^k)| > |\Delta_{h_k}^2 f(x_{i+1}^k)| \quad \text{and} \quad |\Delta_{h_k}^2 f(x_{i-1}^k)| > |\Delta_{h_k}^2 f(x_{i-2}^k)|$$
then I_i^k is initially labelled B.
3. During the interpolation procedure, there is the possibility that a B-interval is relabelled G (see below).

Note that neither detection rule implies the other and that an interval can be labelled B by both rules at the same time. The interpolation procedure is as follows:

1. If I_i^k was labelled as G, then we take the linear interpolation on this interval as approximation for f,
$$\mathcal{I}_i^{h_k} f(x) = N_k(f(x_i^k) - f(x_{i-1}^k))(x - x_{i-1}^k) + f(x_{i-1}^k).$$
2. If I_i^k was labelled as B and the two adjacent intervals as G, we construct linear functions p_{i-1}^k and p_{i+1}^k by linearly interpolating on the intervals I_{i-1}^k and I_{i+1}^k, respectively, and use them to predict the location of the singularity. If both lines intersect at a single point y inside I_i^k, then we define
$$\mathcal{I}_i^{h_k} f(x) = \begin{cases} p_{i-1}^k(x) & x \leqslant y, \\ p_{i+1}^k(x) & x > y, \end{cases}$$
for $x \in I_i^k$. Otherwise we relabel I_i^k as good.
3. If I_i^k belongs to a B-pair, we treat the pair as a single bad interval and return to step 2. This however means that $\mathcal{I}_i^{h_k} f$ does not need to interpolate f at the midpoint of the B-pair.

Lemma 5.1 below guarantees that these are indeed the only possibilities. To conclude, we write the final interpolant as

$$\mathcal{I}^{h_k} f(x) = \sum_{i=1}^{N_k} \mathcal{I}_i^{h_k} f(x) \mathbb{1}_{[x_{i-1}^k, x_i^k)}(x).$$

5.1.2 Properties

In what follows, we present some useful properties of ENO-SR-2. The first two are due to [Aràndiga et al., 2005]. In particular, they guarantee that the described ENO-SR method is second-order accurate for continuous, piecewise smooth functions.

Lemma 5.1 *The groups of adjacent B intervals are at most of size 2.*

Proof See Lemma 1 in [Aràndiga et al., 2005]. □

Theorem 5.2 *Let f be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. The interpolant $\mathcal{I}^{h_k} f$ satisfies*

$$\|f - \mathcal{I}^{h_k} f\|_\infty \leqslant C h_k^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|$$

for all $h_k > 0$, with $C > 0$ independent of f.

Proof See Theorem 1 in [Aràndiga et al., 2005]. □

Similarly to Section 4.3, we wish to develop a method that exactly reproduces the output of ENO-SR-2 using ReLU neural networks. We are amongst others interested in investigating whether there exists a ReLU neural network that maps the input data to the the ENO-SR-2 interpolant evaluated at a fixed grid point. For this to be possible, the ENO-SR-2 procedure should be a continuous mapping, as ReLU neural networks are continuous functions as well. The following lemma ascertains that this is in fact not the case.

Lemma 5.3 *The ENO-SR-2 interpolant does not continuously depend on the input data.*

Proof We prove the lemma by giving an example that shows that $\mathcal{I}^{h_k} f$ does not need to depend continuously on f^k when the second-order ENO-SR algorithm is used. The function values $f^k = \{f(x_i)\}_{i=0}^6$ and the absolute values of the corresponding second-order finite differences are shown in the table below. We wish to find an approximation of $f(x_5^{k+1})$. Note that x_5^{k+1} is the midpoint of the interval $I_3^k = [x_2^k, x_3^k] = [2, 3]$.

$x_i^k = i$	0	1	2	3	4	5	6
$f(x_i^k)$	0	0	0	1	$3+\varepsilon$	$6+2\varepsilon$	$9+3\varepsilon$
$\lvert\Delta_1 f(x_i^k)\rvert$	/	0	1	$1+\varepsilon$	1	0	/

If $\varepsilon > 0$, then I_3^k and I_4^k are labelled bad by both rules. The interpolants used to estimate the discontinuity are therefore $p_{2,\varepsilon}^k(x) = 0$ and $p_{5,\varepsilon}^k(x) = (3+\varepsilon)(x - x_3^k)$. We calculate that the intersection point of $p_{2,\varepsilon}^k$ and $p_{5,\varepsilon}^k$ is $y_{2,5,\varepsilon}^k = x_3^k$. Note that $y_{2,5,\varepsilon}^k \in I_3^k \cup I_4^k$, so that there is no relabelling. Furthermore we have $x_5^{k+1} < y_{2,5,\varepsilon}^k$ such that $\mathcal{I}_{3,\varepsilon}^{h_k} f(x_5^{k+1}) = p_{2,\varepsilon}^k(x_5^{k+1}) = 0$.

If $\varepsilon = 0$, then all intervals are labelled as good. We get that

$$\mathcal{I}_{3,\varepsilon=0}^{h_k} f(x_5^{k+1}) = p_{3,\varepsilon=0}^k(x_5^{k+1}) = \frac{1}{2}.$$

Since $\lim_{\varepsilon\to 0}\mathcal{I}^{h_k}_{3,\varepsilon}f(x^{k+1}_5) \neq \mathcal{I}^{h_k}_{3,\varepsilon=0}f(x^{k+1}_5)$, we have proven that the interpolation procedure induced by the second-order ENO-SR algorithm cannot be written as a continuous function with input f^k and output f^{k+1}. In particular, it cannot be written as a ReLU neural network with continuous output activation function. □

Corollary 5.4 *There is no ReLU neural network with a continuous output function, input f^k and output $(\mathcal{I}^{h_k}f)^{k+1}$ from second-order ENO-SR.*

5.1.3 ENO-SR-2 stencil selection with ReLU neural networks

We adopt the setting of Section 4.1. As in (4.13), the goal is to determine $\mathcal{I}^{h_k}_i(x^{k+1}_{2i-1})$, the interpolation of f^k in x^{k+1}_{2i-1} (the midpoint of I^k_i), for every $1 \leqslant i \leqslant N_k$. From the ENO-SR interpolation procedure it is clear that for every i there exists $r_i \in \{-2,-1,0,1,2\}$ such that $\mathcal{I}^{h_k}_i(x^{k+1}_{2i-1}) = p^k_{i+r_i}(x^{k+1}_{2i-1})$. Analogously to what was described in Section 4.3, this gives rise to a classification problem. Instead of considering the stencil shifts as the output classes of the network, one can also treat the different cases that are implicitly described in the ENO-SR interpolation procedure in Section 5.1.1 as classes. This enables us to construct a ReLU neural network of which the output clearly indicates for each interval which stencil shift should be used. From this, one can easily calculate

$$\mathcal{I}^{h_k}_i(x^{k+1}_{2i-1}) = N_k(f^k_{i+r_i} - f^k_{i+r_i-1})(x^{k+1}_{2i-1} - x^k_{i+r_i-1}) + f^k_{i+r_i-1}.$$

Note that one can also adapt the output activation function of the ANN to include the above, such that for each $1 \leqslant i \leqslant N_k$ the output is $\mathcal{I}^{h_k}_i f(x^{k+1}_{2i-1})$ instead of r_i.

Theorem 5.5 *There exists a ReLU neural network with a suitable output function, input $f^k \in \mathbb{R}^{N_k+1}$ and output $(r_1,\dots,r_{N_k}) \in \mathbb{R}^{N_k}$ as defined above.*

Proof We will refrain from writing down explicitly the exact weights and biases of the ANN. Instead, we will break the ENO-SR-2 algorithm down in different steps, all of which can be written as a ReLU neural network. The basic ReLU calculus results from Section 2.2 then ensure that the composition is again a ReLU neural network. A possible network implementation will be discussed later. In what follows, the vectors X^j do not need to refer to the layers of that specific network implementation. We also assume without loss of generality that $x^k_i = i$ for $0 \leqslant i \leqslant N_k$.

The input of the ANN will be the vector $X^0 \in \mathbb{R}^{N_k+1}$ such that $X^0_{i+1} = f(x^k_i)$ for all $0 \leqslant i \leqslant N_k$. Using a simple affine transformation, we can obtain $X^1 \in \mathbb{R}^{N_k-1}$ such that $X^1_i = \Delta_{h_k} f(x^k_i)$ for all $1 \leqslant i \leqslant N_k - 1$.

We now define the following quantity,

$$M_i = \max_{n=1,2}|\Delta^2_{h_k} f(x^k_{i\pm n})| = \max_{n=1,2}|X^1_{i\pm n}| \tag{5.1}$$

where $3 \leqslant i \leqslant N_k - 3$. Next, we construct a vector $X^2 \in \mathbb{R}^{N_k}$ such that every entry corresponds to an interval. For $1 \leqslant i \leqslant N_k$, we want $X^2_i > 0$ if and only if the interval I^k_i is labelled as B by rule 1 or 2 of the ENO-SR detection mechanism. Good intervals I^k_i will have $X^2_i = 0$ (excluding relabelled G intervals). We can achieve this by defining

$$\begin{aligned} X^2_i =&(\min\{|X^1_i| - |X^1_{i+1}|, |X^1_{i-1}| - |X^1_{i-2}|\})_+ \\ &+ (|X^1_i| - M_i)_+ + (|X^1_{i-1}| - M_{i-1})_+ \end{aligned} \tag{5.2}$$

for $4 \leqslant i \leqslant N_k - 3$. Furthermore we set $X_1^2 = X_2^2 = X_3^2 = X_{N_k-2}^2 = X_{N_k-1}^2 = X_{N_k}^2 = 0$. We thus assume that the discontinuity is far enough from the boundary of the interval. This can be achieved by taking k large enough, or by introducing suitably prescribed ghost values. Note that the first term of the sum will be strictly positive if I_i^k is labelled bad by the second rule of the detection mechanism and one of the other terms will be strictly positive if I_i^k is labelled bad by the first rule.

So far we only considered the first and second rule of the ENO-SR detection mechanism. The third rule states that a bad interval can be relabelled as good during the interpolation procedure. We denote by $p_i(x) = a_i x + b_i$ the linear interpolation on I_i^k. It can be easily seen that

$$\begin{aligned} a_i &= N_k(f(x_i^k) - f(x_{i-1}^k)), \\ b_i &= f(x_{i-1}^k) - N_k(f(x_i^k) - f(x_{i-1}^k))x_{i-1}^k. \end{aligned}$$

A bad interval I_i^k will be relabelled as good in three cases:

- The intervals I_{i-1}^k and I_{i+1}^k were labelled as good and the interpolants p_{i-1} and p_{i+1} do not intersect in a single point of I_i^k.
- The interval I_{i-1}^k was labelled bad, I_{i-2}^k and I_{i+1}^k good and the interpolants p_{i-2} and p_{i+1} do not intersect in a single point of $I_{i-1}^k \cup I_i^k$.
- The interval I_{i+1}^k was labelled bad, I_{i-1}^k and I_{i+2}^k good and the interpolants p_{i-1} and p_{i+2} do not intersect in a single point of $I_i^k \cup I_{i+1}^k$.

The goal is to develop for each case a quantity that is zero when the case is applicable. Let $(m, n) \in \{(i-2, i+1), (i-1, i+1), (i-1, i+2)\}$ like in the cases above. We first consider the case where $a_m = a_n$. In this case the interpolants do not have a single intersection point in I_i^k. Note that $|a_m - a_n|$ is a quantity that is zero when $a_m = a_n$ and strictly larger otherwise.

Second, we treat the case $a_m \neq a_n$. Let $y_{m,n}$ denote the intersection point of the two interpolants coming from the intervals I_m^k and I_n^k, i.e.

$$y_{m,n} = \frac{b_n - b_m}{a_m - a_n}.$$

We want to define a quantity that is equal to zero when $y_{m,n} \in [z_1, z_2]$ and strictly larger otherwise. We first develop a condition to determine whether $|y_{m,n}|$ is left or right from some point z. Define for this goal

$$\gamma_{m,n}(z) = |b_m - b_n| - z|a_m - a_n|. \tag{5.3}$$

We obtain

$$\begin{aligned} |y_{m,n}| < z &\iff \gamma_{m,n}(z) < 0, \\ |y_{m,n}| > z &\iff \gamma_{m,n}(z) > 0. \end{aligned}$$

When we use this notation, we have for example that $|y_{i-1,i+1}| \in I_i^k$ if and only if $\gamma_{i-1,i+1}(x_{i-1}^k) \geqslant 0$ and $\gamma_{i-1,i+1}(x_i^k) \leqslant 0$. It remains to check that $y_{m,n} = |y_{m,n}|$. Define

$$\lambda_{m,n} = \min\{(b_m - b_n)_+, (a_m - a_n)_+\} + \min\{(b_n - b_m)_+, (a_n - a_m)_+\}. \tag{5.4}$$

One can check that

$$\begin{aligned} y_{m,n} < 0 &\iff \lambda_{m,n} > 0, \\ y_{m,n} \geqslant 0 &\iff \lambda_{m,n} = 0. \end{aligned}$$

We can now define

$$\beta_{m,n}(z_1, z_2) = \lambda_{m,n} + (-\gamma_{m,n}(z_1))_+ + (\gamma_{m,n}(z_2))_+ + X^2_m + X^2_n \tag{5.5}$$

and observe that for $0 \leqslant z_1 \leqslant z_2$ the quantity $\beta_{m,n}(z_1, z_2)$ indeed is equal to zero when $y_{m,n} \in [z_1, z_2]$ and the intervals I^k_m and I^k_n were labelled as good. It is strictly larger otherwise.

Now define $n_{i,l} = l + 13(i-1)$ for $1 \leqslant i \leqslant N_k$ and $1 \leqslant l \leqslant 13$. Using this notation, i refers to the interval I^k_i. Define $X^3 \in \mathbb{R}^{13N_k}$ in the following way:

$$\begin{array}{ll} X^3_{n_{i,1}} = X^2_i & X^3_{n_{i,8}} = \beta_{i-1,i+2}(x^{k+1}_{2i-1}, x^k_{i+1}) \\ X^3_{n_{i,2}} = X^2_{i-1} + X^2_{i+1} + |a_{i-1} - a_{i+1}| & X^3_{n_{i,9}} = X^2_{i-1} + X^2_{i+2} \\ X^3_{n_{i,3}} = \beta_{i-1,i+1}(x^k_{i-1}, x^{k+1}_{2i-1}) & X^3_{n_{i,10}} = X^2_{i-2} + X^2_{i+1} + |a_{i-2} - a_{i+1}| \\ X^3_{n_{i,4}} = \beta_{i-1,i+1}(x^{k+1}_{2i-1}, x^k_i) & X^3_{n_{i,11}} = \beta_{i-2,i+1}(x^k_{i-2}, x^{k+1}_{2i-1}) \\ X^3_{n_{i,5}} = X^2_{i-1} + X^2_{i+1} & X^3_{n_{i,12}} = \beta_{i-2,i+1}(x^{k+1}_{2i-1}, x^k_i) \\ X^3_{n_{i,6}} = X^2_{i-1} + X^2_{i+2} + |a_{i-1} - a_{i+2}| & X^3_{n_{i,13}} = X^2_{i-2} + X^2_{i+1} \\ X^3_{n_{i,7}} = \beta_{i-1,i+2}(x^k_{i-1}, x^{k+1}_{2i-1}) & \end{array} \tag{5.6}$$

for $6 \leqslant i \leqslant N_k - 5$. We set $X^3_{n_{i,l}} = 0$ for $1 \leqslant l \leqslant 13$ and $1 \leqslant i \leqslant 5$ or $N_k - 4 \leqslant i \leqslant N_k$. We can now define the output $Y \in \mathbb{R}^{N_k}$ of the ReLU neural network by

$$Y_i = \min \operatorname{argmin}_{1 \leqslant l \leqslant 13} X^3_{n_{i,l}}. \tag{5.7}$$

Table 5.1 shows how one can directly obtain r_i from Y_i. The second column shows the label of interval I^k_i: G refers to intervals which were never labelled B by the detection mechanism, G^* stands for good after relabelling, B means bad. The 'interpolants' column shows the interpolants used to estimate the location of the discontinuity for an initially bad interval. The fifth column tells whether those interpolants intersect, and if so, where.

Note that all possible cases are listed in the table. After the first two steps of the ENO-SR-2 detection mechanism, I^k_i is either given the initial label G or B. The former corresponds to $Y_i = 1$, the latter to all other values of Y_i. As a consequence of Lemma 5.1 there are three options when I^k_i was initially labelled B: it either is an isolated bad interval or it belongs to a bad pair, either with I^k_{i-1} or with I^k_{i+1}. Every of these three cases can then be partitioned into four subcases, depending on the existence of the intersection point of the two interpolants and its location (if applicable).

To conclude the proof, we motivate that the value of Y_i indeed corresponds to the situations described in Table 5.1. We will refer to the description in the table corresponding to $Y_i = l$ as case l. By construction, we know that $X^3_{n_{i,l}} = 0$ if case l is applicable. Conversely, it is possible that $X^3_{n_{i,l}} = 0$ even though case l is not applicable. If for example all intervals are good, then $X^3_{n_{i,1}} = X^3_{n_{i,5}} = X^3_{n_{i,9}} = X^3_{n_{i,13}} = 0$. Or more problematic,

Y_i	label	initially bad pair?	interpolants	single intersection point?	r_i
1	G	/	/	/	0
2	G^*	no	p_{i-1}, p_{i+1}	no, parallel lines	0
3	B	no	p_{i-1}, p_{i+1}	yes, in $[x_{i-1}^k, x_{2i-1}^{k+1}]$	1
4	B	no	p_{i-1}, p_{i+1}	yes, in $[x_{2i-1}^{k+1}, x_i^k]$	-1
5	G^*	no	p_{i-1}, p_{i+1}	yes, but outside I_i^k	0
6	G^*	yes, with I_{i+1}^k	p_{i-1}, p_{i+2}	no, parallel lines	0
7	B	yes, with I_{i+1}^k	p_{i-1}, p_{i+2}	yes, in $[x_{i-1}^k, x_{2i-1}^{k+1}]$	2
8	B	yes, with I_{i+1}^k	p_{i-1}, p_{i+2}	yes, in $[x_{2i-1}^{k+1}, x_{i+1}^k]$	-1
9	G^*	yes, with I_{i+1}^k	p_{i-1}, p_{i+2}	yes, but outside $I_i^k \cup I_{i+1}^k$	0
10	G^*	yes, with I_{i-1}^k	p_{i-2}, p_{i+1}	no, parallel lines	0
11	B	yes, with I_{i-1}^k	p_{i-2}, p_{i+1}	yes, in $[x_{i-2}^k, x_{2i-1}^{k+1}]$	1
12	B	yes, with I_{i-1}^k	p_{i-2}, p_{i+1}	yes, in $[x_{2i-1}^{k+1}, x_i^k]$	-2
13	G^*	yes, with I_{i-1}^k	p_{i-2}, p_{i+1}	yes, but outside $I_{i-1}^k \cup I_i^k$	0

Table 5.1: Overview of all possible values of Y_i and how they relate to r_i. Further explanation about the columns can be found in the main text.

it is possible that $X_{n_i,3}^3 = 0$ even though I_i^k is a good interval, which corresponds to a different value of r_i. However, thanks to the definition of X^3, in particular the specific order of its entries, the correspondence as in Table 5.1 is justified. Indeed, if $Y_i \geqslant 2$ then I_i^k must have been initially labelled bad. Furthermore, if $Y_i \in \{3, 4\}$ and thus $Y_i \neq 2$ then $|a_{i-1} - a_{i+1}| \neq 0$ such that we are justified to use definition (5.5). Similarly, if $Y_i \geqslant 6$ then I_i^k must belong to a bad pair and if $Y_i \geqslant 10$ then (I_{i-1}^k, I_i^k) must be the bad pair. This concludes the proof of the theorem. □

Now that we have proven that ENO-SR-2 can be written as a ReLU neural network with a suitable output function, we present a possible architecture of such an ANN that calculates Y_i from f^k. The network we present has four hidden layers, of which the width varies from 29 to 39, and an output layer of 13 neurons. It is visualized using a flowchart in Figure 5.1. The entire network to calculate Y can then easily be obtained by concatenating these local networks. With more effort, one can also take into account that adjacent local networks share quite a lot of neurons, leading to a reduction of the width of the network.

We recall the following observations from Section 2.2:

$$a = (a)_+ - (-a)_+, \tag{5.8a}$$

$$|a| = (a)_+ + (-a)_+, \tag{5.8b}$$

$$\max\{a, b\} = a + (b - a)_+, \tag{5.8c}$$

$$\min\{a, b\} = a - (a - b)_+, \tag{5.8d}$$

$$\max\{a, b, c, d\} = a + (b - a)_+ + \big(c + (d - c)_+ - a - (b - a)_+\big)_+, \tag{5.8e}$$

for all $a, b, c, d \in \mathbb{R}$. Furthermore we introduce the shorthand notation $\Delta_q = \Delta_{h_k}^2 f(x_{i+q}^k)$, where we keep i and k are fixed during the remainder of this section. To simplify notation, the flowchart shows the values on the neurons before the activation function is applied.

We now give some more explanation about how all the layers can be calculated from the previous layer in Figure 5.1. The bold letters in this paragraph refer to the letters next

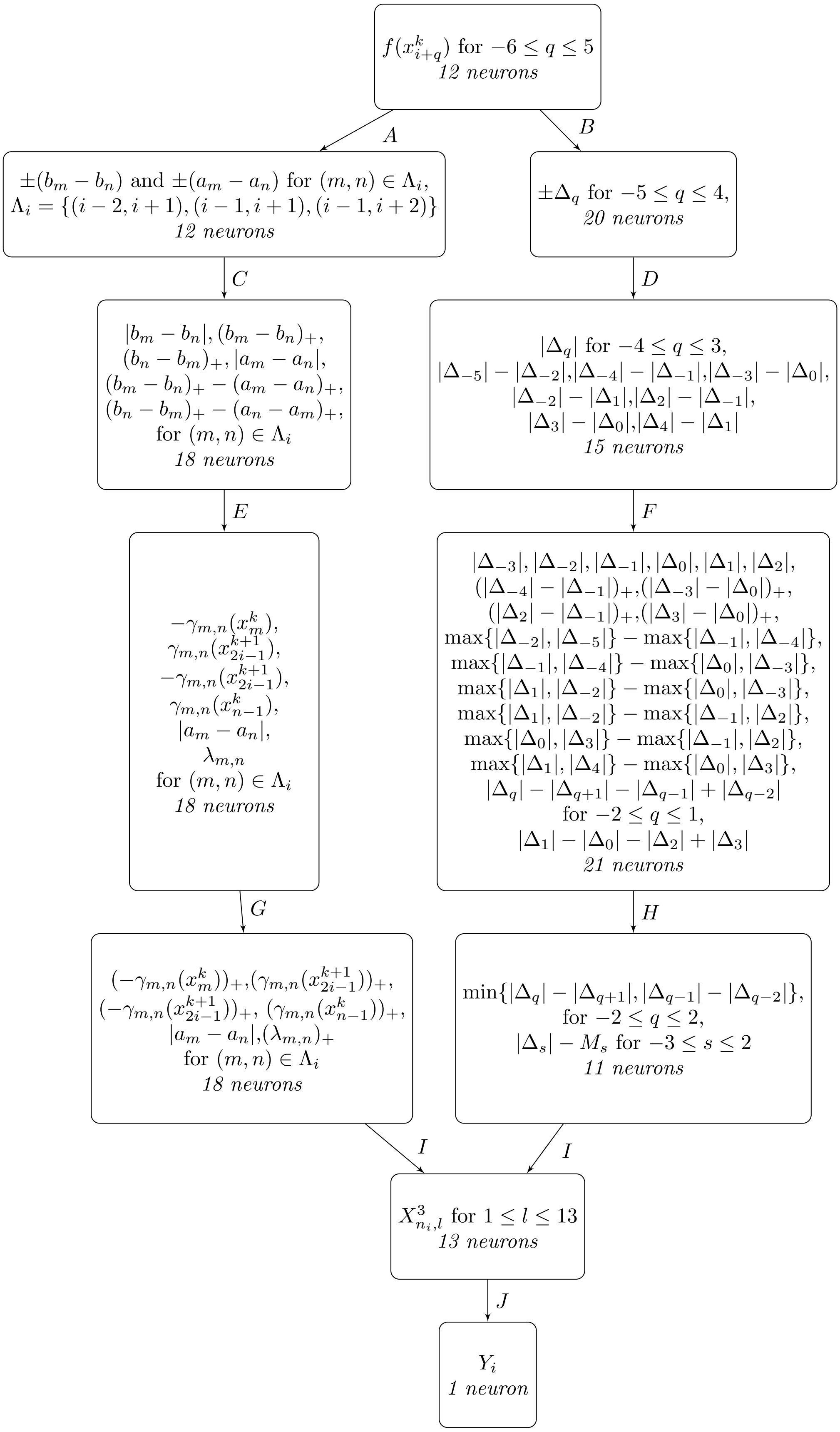

$f(x^k_{i+q})$ for $-6 \leq q \leq 5$
12 neurons
A
B
$\pm(b_m - b_n)$ and $\pm(a_m - a_n)$ for $(m,n) \in \Lambda_i$,
$\Lambda_i = \{(i-2,i+1),(i-1,i+1),(i-1,i+2)\}$
12 neurons
$\pm\Delta_q$ for $-5 \leq q \leq 4$,
20 neurons
C
D
$|b_m - b_n|, (b_m - b_n)_+$,
$(b_n - b_m)_+, |a_m - a_n|$,
$(b_m - b_n)_+ - (a_m - a_n)_+$,
$(b_n - b_m)_+ - (a_n - a_m)_+$,
for $(m,n) \in \Lambda_i$
18 neurons
$|\Delta_q|$ for $-4 \leq q \leq 3$,
$|\Delta_{-5}| - |\Delta_{-2}|$,$|\Delta_{-4}| - |\Delta_{-1}|$,$|\Delta_{-3}| - |\Delta_0|$,
$|\Delta_{-2}| - |\Delta_1|$,$|\Delta_2| - |\Delta_{-1}|$,
$|\Delta_3| - |\Delta_0|$,$|\Delta_4| - |\Delta_1|$
15 neurons
E
F
$-\gamma_{m,n}(x^k_m)$,
$\gamma_{m,n}(x^{k+1}_{2i-1})$,
$-\gamma_{m,n}(x^{k+1}_{2i-1})$,
$\gamma_{m,n}(x^k_{n-1})$,
$|a_m - a_n|$,
$\lambda_{m,n}$
for $(m,n) \in \Lambda_i$
18 neurons
$|\Delta_{-3}|, |\Delta_{-2}|, |\Delta_{-1}|, |\Delta_0|, |\Delta_1|, |\Delta_2|$,
$(|\Delta_{-4}| - |\Delta_{-1}|)_+$,$(|\Delta_{-3}| - |\Delta_0|)_+$,
$(|\Delta_2| - |\Delta_{-1}|)_+$,$(|\Delta_3| - |\Delta_0|)_+$,
$\max\{|\Delta_{-2}|, |\Delta_{-5}|\} - \max\{|\Delta_{-1}|, |\Delta_{-4}|\}$,
$\max\{|\Delta_{-1}|, |\Delta_{-4}|\} - \max\{|\Delta_0|, |\Delta_{-3}|\}$,
$\max\{|\Delta_1|, |\Delta_{-2}|\} - \max\{|\Delta_0|, |\Delta_{-3}|\}$,
$\max\{|\Delta_1|, |\Delta_{-2}|\} - \max\{|\Delta_{-1}|, |\Delta_2|\}$,
$\max\{|\Delta_0|, |\Delta_3|\} - \max\{|\Delta_{-1}|, |\Delta_2|\}$,
$\max\{|\Delta_1|, |\Delta_4|\} - \max\{|\Delta_0|, |\Delta_3|\}$,
$|\Delta_q| - |\Delta_{q+1}| - |\Delta_{q-1}| + |\Delta_{q-2}|$
for $-2 \leq q \leq 1$,
$|\Delta_1| - |\Delta_0| - |\Delta_2| + |\Delta_3|$
21 neurons
G
H
$(-\gamma_{m,n}(x^k_m))_+$,$(\gamma_{m,n}(x^{k+1}_{2i-1}))_+$,
$(-\gamma_{m,n}(x^{k+1}_{2i-1}))_+$, $(\gamma_{m,n}(x^k_{n-1}))_+$,
$|a_m - a_n|$,$(\lambda_{m,n})_+$
for $(m,n) \in \Lambda_i$
18 neurons
$\min\{|\Delta_q| - |\Delta_{q+1}|, |\Delta_{q-1}| - |\Delta_{q-2}|\}$,
for $-2 \leq q \leq 2$,
$|\Delta_s| - M_s$ for $-3 \leq s \leq 2$
11 neurons
I
I
$X^3_{n_i,l}$ for $1 \leq l \leq 13$
13 neurons
J
Y_i
1 neuron

Figure 5.1: Flowchart of a ReLU ANN to calculate the output of the ENO-SR-2 procedure Y_i from f^k.

to the arrows in the flowchart. We will refer to the input layer as the zeroth layer, and so on. ***A.B.*** It is easy to see that all quantities of the first layer are linear combinations of the input neurons. ***C.*** Straightforward application of (5.8b) on $\pm(b_m - b_n)$ and $\pm(a_m - a_n)$ and some linear combinations. ***D.*** Straightforward application of (5.8b) on $\pm\Delta_q$, followed by taking linear combinations. ***E.*** Note that the ReLU activation does not affect the first four quantities of the third layer in the left subnetwork as they are positive. From the definitions of $\gamma_{m,n}$ and $\lambda_{m,n}$ in equations (5.3) and (5.4), respectively, it is clear that this is again nothing more than the straightforward calculation of linear combinations and the application of (5.8d). ***F.*** The first ten quantities were forwarded from the previous layer. The next six are applications of (5.8c), where the order of the arguments of the maximums are carefully chosen. The last five are linear combinations of the quantities of the third layer. ***G.*** Forwarding values. ***H.*** For the calculation of the minimum, (5.8d) can be used. Note that for $q = 2$ it was used that $\min\{a, b\} = b - (b - a)_+$. For the calculation of M_s, recall equations (5.1) and (5.8e). ***I.*** The result follows from combining definitions (5.2), (5.5), (5.6) and (5.8c). ***J.*** As can be seen in definition (5.7), Y_i is obtained by applying the output activation function min argmin on the output layer.

5.2 Adapted second-order ENO-SR interpolation

Based on the second-order ENO-SR procedure, as proposed by [Aràndiga et al., 2005] and described in Section 5.1.1, we propose an adapted ENO-SR procedure that can be more easily recast as a ReLU neural network, without losing second-order accuracy. In the following, we describe this adapted procedure and prove that it indeed is still second-order accurate. Next, we present a new version of Theorem 5.5, in which the number of classes is reduced from thirteen to four. Similarly to Section 4.4, we approach the ENO-SR procedure again as a regression problem instead of classification problem.

5.2.1 Algorithm

The first step of the adapted procedure is again to label intervals that might contain the singular point z as B (bad), other intervals get the label G (good). We use again second-order differences

$$\Delta_h^2 f(x) := f(x-h) - 2f(x) + f(x+h) \tag{5.9}$$

as smoothness indicators. The rules of the adapted ENO-SR detection mechanism are the following:

1. The intervals I_{i-1}^k and I_i^k are labelled B if
$$|\Delta_{h_k}^2 f(x_{i-1}^k)| > \max_{n=1,2,3} |\Delta_{h_k}^2 f(x_{i-1\pm n}^k)|.$$

2. Interval I_i^k is labelled B if
$$|\Delta_{h_k}^2 f(x_i^k)| > \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i+n}^k)| \quad \text{and} \quad |\Delta_{h_k}^2 f(x_{i-1}^k)| > \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i-1-n}^k)|.$$

3. All other intervals are labelled G.

Note that neither detection rule implies the other and that an interval can be labelled B by both rules at the same time. In the following, we denote by $p_i^k : [c,d] \to \mathbb{R} : x \mapsto a_i x + b_i$ the linear interpolation of the endpoints of I_i^k, where we write a_i and b_i instead of a_i^k and b_i^k to simplify notation. The interpolation procedure is as follows:

1. If I_i^k was labelled as G, then we take the linear interpolation on this interval as approximation for f,
$$\mathcal{I}_i^{h_k} f(x) = p_i^k(x).$$

2. If I_i^k was labelled as B, we use p_{i-2}^k and p_{i+2}^k to predict the location of the singularity. If both lines intersect at a single point y , then we define
$$\mathcal{I}_i^{h_k} f(x) = \begin{cases} p_{i-2}^k(x) & x \leqslant y, \\ p_{i+2}^k(x) & x > y, \end{cases}$$
Otherwise we treat I_i^k as a good interval and let $\mathcal{I}_i^{h_k} f(x) = p_i^k(x)$.

We then define the final interpolant by requiring that the restriction of $\mathcal{I}^{h_k} f$ to the interval I_i^k is equal to $\mathcal{I}_i^{h_k} f$.

5.2.2 Properties

We present some properties of our adaptation of the ENO-SR algorithm. To prove the second-order accuracy, we state some results due to [Aràndiga et al., 2005] in a slightly adapted form.

Lemma 5.6 *The groups of adjacent B intervals are at most of size 2. They are separated by groups of adjacent G intervals that are at least of size 2.*

Proof Note that our detection algorithm is the same as the one in [Aràndiga et al., 2005] for $m = 3$. The result then follows from their Lemma 1. □

Lemma 5.7 *Let f be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. Define the critical scale*
$$h_c := \frac{|[f']|}{4 \sup_{x\in\mathbb{R}\backslash\{z\}} |f''(x)|}, \tag{5.10}$$
where $[f']$ is the jump of the first derivative f' at the point z. Then for $h < h_c$, the interval that contains z is labelled B.

Proof See Lemma 2 in [Aràndiga et al., 2005]. □

Lemma 5.8 *There exist constants $C > 0$ and $0 < K < 1$ such that for all continuous f with uniformly bounded second derivative on $\mathbb{R}\backslash\{z\}$ and for $h < K h_c$ with h_c defined by (5.10), the following holds:*

1. *The singularity z is contained in an isolated B interval I_i^k or in a B-pair (I_i^k, I_{i+1}^k).*
2. *The two polynomials p_{i-2}^k and p_{i+2}^k (or p_{i-1}^k and p_{i+3}^k) have only one intersection point y inside I_i^k or $I_i^k \cup I_{i+1}^k$, respectively.*

3. The distance between z and y is bounded by

$$|z-y| \leqslant \frac{C \sup_{x\in\mathbb{R}\backslash\{z\}} |f''(x)| h^2}{|[f']|}. \tag{5.11}$$

Proof This is a light adaptation of Lemma 3 in [Aràndiga et al., 2005]. The proof remains the same, after one minor change. We take $I = [b, c]$ to be equal to $I_{-1} \cup I_0 \cup I_1$, which does not affect equation (38) in the proof. In fact, all other steps of their proof remain valid. It only must be noted that the constant C in equation (5.11) of this thesis and equation (37) in [Aràndiga et al., 2005] do not necessarily agree. □

The theorem below states that our adaptation of ENO-SR is indeed second-order accurate and can also be found in [De Ryck et al., 2020].

Theorem 5.9 *Let f be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. The adapted ENO-SR interpolant $\mathcal{I}^h f$ satisfies*

$$\|f - \mathcal{I}^h f\|_\infty \leqslant Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|$$

for all $h > 0$, with $C > 0$ independent of f.

Proof This theorem and its proof are based on Theorem 1 in [Aràndiga et al., 2005]. Let h_c be as in Lemma 5.7 and let K be as in Lemma 5.8. Note that we can write

$$f = f_- \mathbb{1}_{(-\infty,z]} + f_+ \mathbb{1}_{(z,+\infty)}$$

where f_-, f_+ are C^2 on $\mathbb{R}$ such that

$$\sup_{\mathbb{R}\backslash\{z\}} |f''_\pm| \leqslant \sup_{\mathbb{R}\backslash\{z\}} |f''|.$$

Let us consider some interval $I_0 = [b, c] = [b, b+h]$. First consider the case $0 < h < Kh_c$. Suppose it was labelled as good. Lemma 5.7 then guarantees that I_0 does not contain z. It then follows directly from the theory of Lagrange interpolation that

$$|f(x) - \mathcal{I}^h f(x)| \leqslant Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''| \tag{5.12}$$

for all $x \in I_0$. Now suppose that I_0 was labelled bad. As a consequence of Lemma 5.6, I_{-2} and I_2 are good intervals and therefore do not contain the discontinuity. If in addition $z \notin I_{-1} \cup I_0 \cup I_1$, then (5.12) holds again for all $x \in I_0$ since $\mathcal{I}^h f(x)$ is either equal to $p_{-2}(x)$, $p_0(x)$ or $p_2(x)$. On the other hand, if $z \in I_{-1} \cup I_0 \cup I_1$ then Lemma 5.8 guarantees the existence of a single intersection point $y \in I_{-1} \cup I_0 \cup I_1$ of p_{-2} and p_2. Assume now without loss of generality that $z \leqslant y$. In this case, equation (5.12) holds for all $x \in [b, z] \cup [y, c]$. It thus remains to treat the case $z < x < y$. For such x, we have

$$|f(x) - \mathcal{I}^h f(x)| = |f_+(x) - p_{-2}(x)| \leqslant |f_+(x) - f_-(x)| + |f_-(x) - p_{-2}(x)|$$

where the second term is again bounded by $Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|$. We can use a second-order Taylor expansion for the first term to derive

$$|f_+(x) - f_-(x)| \leqslant (y-z)([f'] + 2h \sup_{\mathbb{R}\backslash\{z\}} |f''|) \leqslant \frac{3}{2} |[f']| (y-z)$$

where in the last inequality we used that $h < h_c$. By invoking the third part of Lemma 5.8, we find indeed that (5.12) holds again. This concludes the proof for the case $h < Kh_c$.

Now suppose that $h \geqslant Kh_c$. First define

$$f_2(x) = f(x) - [f'](x-z)_+$$

for $x \in \mathbb{R}$. Furthermore, by the definition of h_c in Lemma 5.7, we find that for $h \geqslant Kh_c$,

$$[f'] = 4h_c \sup_{\mathbb{R}\backslash\{z\}} |f''| \leqslant C_0 h \sup_{\mathbb{R}\backslash\{z\}} |f''|, \tag{5.13}$$

where $C_0 > 0$ does not depend on f. We distinguish two cases.

Case 1: $\mathcal{I}^h(x) = p_0(x)$ for all $x \in I_0$. If $z \notin I_0$, second-order accuracy as in (5.12) is immediate. If not, more work is needed. Define

$$g_1(x) = \frac{[f'](x_0 - z)_+}{h}(x - x_{-1})$$

and note that $p_0 - g_1$ is the linear interpolation between $(x_{-1}, f_2(x_{-1}))$ and $(x_0, f_2(x_0))$. Since f_2 is C^2 we know that $p_0 - g_1$ is a second-order accurate approximation of f_2 on I_0, such that (5.12) holds. We then calculate for $x \in I_0$,

$$\begin{aligned}
|f(x) - p_0(x)| &\leqslant |f_2(x) - (p_0(x) - g_1(x))| + |[f'](x-z)_+ - g_1(x)| \\
&\leqslant C_1 h^2 \sup_{\mathbb{R}\backslash\{z\}} |f''| + [f']\left(|(x-z)_+| + \frac{(x_0 - z)_+}{h}|x - x_{-1}|\right) \\
&\leqslant C_1 h^2 \sup_{\mathbb{R}\backslash\{z\}} |f''| + C_0 h \sup_{\mathbb{R}\backslash\{z\}} |f''|(h+h) \\
&= Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|,
\end{aligned}$$

where we used (5.13).

Case 2: $\mathcal{I}^h(x) = p_{-2}(x)\mathbb{1}_{(-\infty,y]}(x) + p_2(x)\mathbb{1}_{(y,+\infty)}(x)$ for $x \in I_0$, where y is the intersection point of p_{-2} and p_2. If $z \notin \cup_{q=-2}^{2} I_q$, then (5.12) follows immediately for $x \in I_0$. Consider now the case that $z \in \cup_{q=-2}^{2} I_q$ and assume without loss of generality $z \leqslant y$. Let $x \in I_0$ be arbitrary. It follows that (5.12) also holds immediately for this x if $y \leqslant x$ or $x \leqslant z$. It suffices to find a bound for when $x_{-3} \leqslant z \leqslant x \leqslant y$. Define

$$g_2(x) = \frac{[f'](x_{-2} - z)_+}{h}(x - x_{-3}).$$

Note that $p_{-2} - g_2$ is an affine function through $(x_{-3}, f_2(x_{-3}))$ and $(x_{-2}, f_2(x_{-2}))$. It follows that

$$\begin{aligned}
|f(x) - p_{-2}(x)| &\leqslant |f_2(x) - (p_{-2} - g_2(x))| + |[f'](x-z)_+ - g_2(x)| \\
&\leqslant C_1 h^2 \sup_{\mathbb{R}\backslash\{z\}} |f''| + [f']\left(|(x-z)_+| + \frac{(x_{-2} - z)_+}{h}|x - x_{-3}|\right) \\
&\leqslant C_1 h^2 \sup_{\mathbb{R}\backslash\{z\}} |f''| + C_0 h \sup_{\mathbb{R}\backslash\{z\}} |f''|(3h+3h) \\
&= Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|,
\end{aligned}$$

where we used again (5.13) and the bounds $|x - x_{-3}| \leqslant 3h$ and $x_{-3} \leqslant z$. This concludes the proof of Theorem 5.9. □

5.2.3 Adapted ENO-SR-2 stencil selection with ReLU neural networks

Assume the setting of Section 4.1. We will now prove that the second-order accurate prediction of f^{k+1} from the previous subsection can be obtained given f^k using a ReLU DNN. The proof we present can be directly generalized to interpolation at points other than the midpoints of the cells, e.g. retrieving cell boundary values for reconstruction purposes. Equation (4.13) shows that we only need to calculate $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ for every $1 \leqslant i \leqslant N_k$. From the ENO-SR interpolation procedure it is clear that for every i there exists $r_i^k \in \{-2, 0, 2\}$ such that $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) = p_{i+r_i^k}^k(x_{2i-1}^{k+1})$. It is thus straightforward to calculate $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ from r_i^k. The following result can also be found in [De Ryck et al., 2020].

Theorem 5.10 *There exists a ReLU neural network with a discontinuous output function, input f^k and output $(r_1^k, \dots, r_{N_k}^k)$ as defined above.*

Proof Instead of explicitly constructing a ReLU neural network, we will prove that we can write the output vector as a composition of functions that can be written as pure ReLU neural networks with linear output functions. Such functions include the rectifier function, absolute value, maximum and the identity function (cfr. Section 2.2). The network architecture of a possible realisation of the network of this proof can be found after the proof. Furthermore we will assume that the discontinuity is not located in the first four or last four intervals. This can be achieved by taking k large enough, or by introducing suitably prescribed ghost values. We also assume without loss of generality that $x_i^k = i$ for $0 \leqslant i \leqslant N_k$.

The input of the neural network will be the vector $X^0 \in \mathbb{R}^{N_k+1}$ with $X_{i+1}^0 = f(x_i^k)$ for all $0 \leqslant i \leqslant N_k$. Using a simple affine transformation, we can obtain $X^1 \in \mathbb{R}^{N_k-1}$ such that $X_i^1 = \Delta_{h_k}^2 f(x_i^k)$ for all $1 \leqslant i \leqslant N_k - 1$. We now define the following quantities,

$$\begin{aligned} M_i &= \max_{n=1,2,3} |\Delta_{h_k}^2 f(x_{i\pm n}^k)| = \max_{n=1,2,3} |X_{i\pm n}^1|, \\ N_i^{\pm} &= \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i\pm n}^k)| = \max_{n=1,2} |X_{i\pm n}^1|, \end{aligned} \tag{5.14}$$

where $4 \leqslant i \leqslant N_k - 4$. Next, we construct a vector $X^2 \in \mathbb{R}^{N_k}$ such that every entry corresponds to an interval. For $1 \leqslant i \leqslant N_k$, we want $X_i^2 > 0$ if and only if the interval I_i^k is labelled as B by the adapted ENO-SR detection mechanism. We can achieve this by defining

$$X_i^2 = (\min\{|X_i^1| - N_i^+, |X_{i-1}^1| - N_{i-1}^-\})_+ + (|X_i^1| - M_i)_+ + (|X_{i-1}^1| - M_{i-1})_+ \tag{5.15}$$

for $5 \leqslant i \leqslant N_k - 4$. Furthermore we set $X_1^2 = X_2^2 = X_3^2 = X_4^2 = X_{N_k-3}^2 = X_{N_k-2}^2 = X_{N_k-1}^2 = X_{N_k}^2 = 0$. Note that the first term of the sum will be strictly positive if I_i^k is labelled bad by the second rule of the detection mechanism and one of the other terms will be strictly positive if I_i^k is labelled bad by the first rule. Good intervals I_i^k have $X_i^2 = 0$.

Now define $n_{i,l} = l + 4(i-1)$ for $1 \leqslant i \leqslant N_k$ and $1 \leqslant l \leqslant 4$. Using this notation, i refers to the interval I_i^k. Define $X^3 \in \mathbb{R}^{4N_k}$ in the following manner:

$$\begin{aligned} X_{n_{i,1}}^3 &= X_i^2, & X_{n_{i,3}}^3 &= \Big(|b_{i-2} - b_{i+2}| - x_{2i-1}^{k+1}|a_{i-2} - a_{i+2}|\Big)_+, \\ X_{n_{i,2}}^3 &= |a_{i-2} - a_{i+2}|, & X_{n_{i,4}}^3 &= \Big(-|b_{i-2} - b_{i+2}| + x_{2i-1}^{k+1}|a_{i-2} - a_{i+2}|\Big)_+, \end{aligned} \tag{5.16}$$

for $5 \leqslant i \leqslant N_k - 4$. We set $X^3_{n_{i,l}} = 0$ for $1 \leqslant l \leqslant 4$ and $1 \leqslant i \leqslant 4$ or $N_k - 3 \leqslant i \leqslant N_k$. We can now define the output $\hat{Y} \in \mathbb{R}^{N_k}$ of the ReLU neural network by

$$\hat{Y}_i = \min \operatorname{argmin}_{1 \leqslant l \leqslant 4} X^3_{n_{i,l}}. \tag{5.17}$$

It remains to prove that r_i^k can be obtained from $\hat{Y}_i$. Note that $\hat{Y}_i = 1$ if and only if I_i^k was labelled G. Therefore $\hat{Y}_i = 1$ corresponds to $r_i^k = 0$. If $\hat{Y}_i = 2$, then I_i^k was labelled B and the interpolants p_{i-2}^k and p_{i+2}^k do not intersect, leading to $r_i^k = 0$ according to the interpolation procedure. Next, $\hat{Y}_i = 3, 4$ corresponds to the case where I_i^k was labelled B and the interpolants p_{i-2}^k and p_{i+2}^k do intersect. This intersection point is seen to be

$$y = \frac{b_{i+2} - b_{i-2}}{a_{i-2} - a_{i+2}}.$$

If $\hat{Y}_i = 3$, then x^{k+1}_{2i-1} is right of y and therefore $r_i^k = 2$. Analogously, $\hat{Y}_i = 4$ corresponds to $r_i^k = -2$, which concludes the proof. □

Now that we have established that the adapted second-order ENO-SR algorithm can also be written as a ReLU ANN with a discontinuous output activation function, we present a possible architecture of an ANN that calculates Y_i from f^k. The network we present has five hidden layers, of which the width varies from 6 to 20, and an output layer of 4 neurons. It is visualized as a flowchart in Figure 5.2. Note that is is a considerably smaller network than the one presented in Figure 5.1.

We now give some more explanation about how all the layers can be calculated from the previous layer in Figure 5.2, where we use the same notation as in the previous section. ***A.B.*** It is easy to see that all quantities of the first layer are linear combinations of the input neurons. ***C.*** Straightforward application of (5.8b) and Definition (5.3) on $\pm(b_{i-2} - b_{i+2})$ and $\pm(a_{i-2} - a_{i+2})$. ***D.*** Straightforward application of (5.8b) on $\pm\Delta_q$, followed by taking linear combinations. ***E.G.I.*** Passing by values. ***F.*** The first six quantities were passed by from the previous layer. The other ones are applications of (5.8c), where the order of the arguments of the maximums is carefully chosen. ***H.*** For the calculation of the minimum, (5.8d) can be used. Furthermore (5.8e) was used. ***J.*** Application of (5.14). ***K.*** The result follows from combining definitions (5.15) and (5.16). ***L.*** As can be seen in definition (5.17), Y_i is obtained by applying the output activation function min argmin on the output layer.

5.2.4 Adapted ENO-SR-2 interpolation with ReLU neural networks

In this subsection, we investigate whether it is possible to extend the previously obtained stencil selection network to a ReLU neural network that maps f^k to $(\mathcal{I}^{h_k} f)^{k+1}$. Recall that Corollary 5.4 proved that there is no such network for the ENO-SR-2 interpolation procedure. The counterexample given in the proof still holds for the adapted ENO-SR-2 procedure, which proves that there is no pure ReLU network with input f^k and output $(\mathcal{I}^{h_k} f)^{k+1}$. In what follows, we design an approximate ENO-SR method, based on the adapted ENO-SR-2 method of the previous section, and investigate its accuracy.

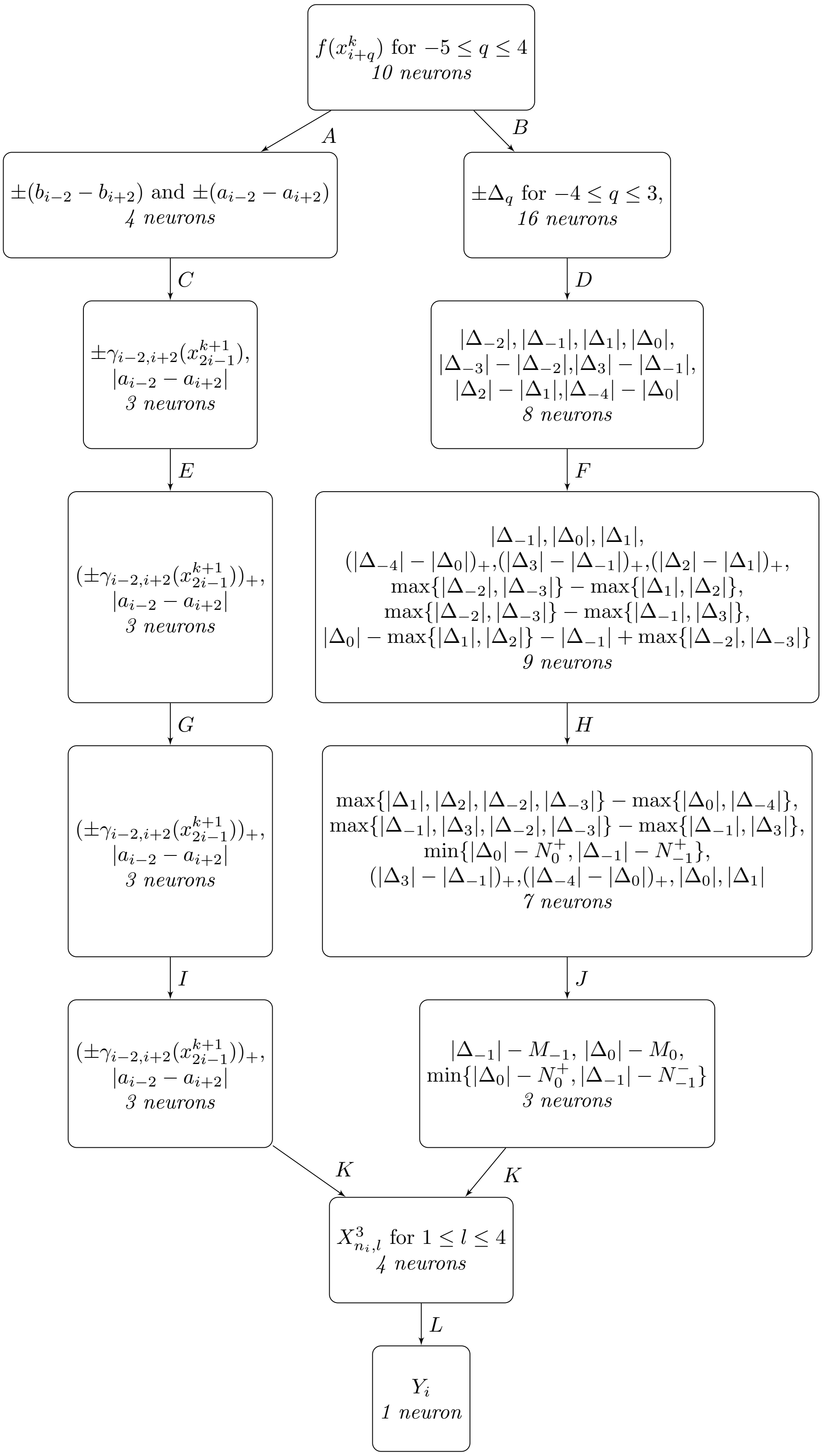

$f(x^k_{i+q})$ for $-5 \leq q \leq 4$
10 neurons
A
B
$\pm(b_{i-2} - b_{i+2})$ and $\pm(a_{i-2} - a_{i+2})$
4 neurons
$\pm\Delta_q$ for $-4 \leq q \leq 3$,
16 neurons
C
D
$\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1})$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-2}|, |\Delta_{-1}|, |\Delta_1|, |\Delta_0|$,
$|\Delta_{-3}| - |\Delta_{-2}|$,$|\Delta_3| - |\Delta_{-1}|$,
$|\Delta_2| - |\Delta_1|$,$|\Delta_{-4}| - |\Delta_0|$
8 neurons
E
F
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-1}|, |\Delta_0|, |\Delta_1|$,
$(|\Delta_{-4}| - |\Delta_0|)_+$,$(|\Delta_3| - |\Delta_{-1}|)_+$,$(|\Delta_2| - |\Delta_1|)_+$,
$\max\{|\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_1|, |\Delta_2|\}$,
$\max\{|\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_{-1}|, |\Delta_3|\}$,
$|\Delta_0| - \max\{|\Delta_1|, |\Delta_2|\} - |\Delta_{-1}| + \max\{|\Delta_{-2}|, |\Delta_{-3}|\}$
9 neurons
G
H
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$\max\{|\Delta_1|, |\Delta_2|, |\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_0|, |\Delta_{-4}|\}$,
$\max\{|\Delta_{-1}|, |\Delta_3|, |\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_{-1}|, |\Delta_3|\}$,
$\min\{|\Delta_0| - N^+_0, |\Delta_{-1}| - N^+_{-1}\}$,
$(|\Delta_3| - |\Delta_{-1}|)_+$,$(|\Delta_{-4}| - |\Delta_0|)_+$, $|\Delta_0|, |\Delta_1|$
7 neurons
I
J
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-1}| - M_{-1}$, $|\Delta_0| - M_0$,
$\min\{|\Delta_0| - N^+_0, |\Delta_{-1}| - N^-_{-1}\}$
3 neurons
K
K
$X^3_{n_i,l}$ for $1 \leq l \leq 4$
4 neurons
L
Y_i
1 neuron

Figure 5.2: Flowchart of a ReLU ANN to calculate the output Y_i of the adapted ENO-SR-2 procedure from f^k.

We recall the function $H_\varepsilon : \mathbb{R} \to \mathbb{R}$, where $\varepsilon \geqslant 0$, defined by

$$H_\varepsilon(x) = \begin{cases} 0 & x \leqslant 0 \\ x/\varepsilon & 0 < x \leqslant \varepsilon \\ 1 & x < \varepsilon. \end{cases} \tag{5.18}$$

Using this function and the notation of the proof of Theorem 5.10, we can write down a single formula for $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$,

$$\begin{aligned} \mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) = (1-\alpha) p_i^k(x_{2i-1}^{k+1}) + \alpha \left((1-\beta) p_{i+2}^k(x_{2i-1}^{k+1}) + \beta p_{i-2}^k(x_{2i-1}^{k+1}) \right), \\ \text{where } \alpha = H_0(\min\{X_{n_{i,1}}^3, X_{n_{i,2}}^3\}), \quad \beta = H_0(X_{n_{i,3}}^3), \end{aligned} \tag{5.19}$$

for $1 \leqslant i \leqslant N_k$. This formula cannot be calculated using a pure ReLU DNN. Nevertheless, we will base ourselves on this formula to introduce an approximate ENO-SR algorithm that can be exactly written as a ReLU DNN.

Similar to what we did in Section 4.4, the first step is to replace H_0 by H_ε with $\varepsilon > 0$, since

$$H_\varepsilon(x) = \frac{1}{\varepsilon}(x)_+ - \frac{1}{\varepsilon}(x-\varepsilon)_+, \tag{5.20}$$

Next we replace the multiplication $\cdot$ by the operation $\star$ as defined in Definition 4.6. For the moment, we assume that there exists $\lambda > 0$ such that all quantities that we will need to multiply, lie in the interval $[-\lambda, \lambda]$. In view of the third property in Lemma 4.7, directly replacing all multiplications in (5.19) by the operation $\star$ will lead to a quantity that is no longer a convex combination of $p_{i-2}^k(x_{2i-1}^{k+1}), p_i^k(x_{2i-1}^{k+1})$ and $p_{i+2}^k(x_{2i-1}^{k+1})$. We therefore introduce the *approximate ENO-SR* prediction $\hat{f}_{i,\varepsilon}^{k+1}$ of $f(x_i^{k+1})$ by setting $\hat{f}_{2i,\varepsilon}^{k+1} = f_{2i}^{k+1}$ for $0 \leqslant i \leqslant N_k$ and

$$\begin{aligned} \hat{f}_{2i-1,\varepsilon}^{k+1} = p_i^k(x_{2i-1}^{k+1}) + \alpha \star \left(p_{i+2}^k(x_{2i-1}^{k+1}) - p_i^k(x_{2i-1}^{k+1}) + \beta \star \left(p_{i-2}^k(x_{2i-1}^{k+1}) - p_{i+2}^k(x_{2i-1}^{k+1}) \right) \right), \\ \text{where } \alpha = H_\varepsilon(\min\{X_{n_{i,1}}^3, X_{n_{i,2}}^3\}), \quad \beta = H_\varepsilon(X_{n_{i,3}}^3), \end{aligned} \tag{5.21}$$

for $1 \leqslant i \leqslant N_k$. The fourth property of Lemma 4.7 ensures that the two convex combinations in (5.19) are replaced by convex combinations (with possibly different weights). The theorem below quantifies the accuracy of the approximate ENO-SR predictions for $\varepsilon > 0$.

Theorem 5.11 *Let $f : [c,d] \to [-1,1]$ be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. For every k, the approximate ENO-SR predictions $\hat{f}_{i,\varepsilon}^{k+1}$ satisfy for every $0 \leqslant i \leqslant N_{k+1}$ and $\varepsilon \geqslant 0$ that*

$$|\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) - \hat{f}_{i,\varepsilon}^{k+1}| \leqslant C h_k^2 \sup_{[c,d]\backslash\{z\}} |f''| + \frac{3}{2}\varepsilon,$$

where $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ is the ENO-SR-2 prediction.

Proof In what follows, we let $x^* = x_{2i-1}^{k+1}$, $f^* = \mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$, $\hat{f}_\varepsilon = \hat{f}_{2i-1,\varepsilon}^{k+1}$ and $X_l^3 = X_{n_{i,l}}^3$ for $1 \leqslant l \leqslant 4$ (where $X_{n_{i,l}}^3$ is as in the proof of Theorem 5.10). We assume without loss of generality that $[c,d] \subset [0,\infty)$. Furthermore we simplify the notation by dropping the index k and setting $i=0$. It follows from the proof of Theorem 5.9 that the results holds for $h \geqslant Kh_c$, since $\hat{f}_\varepsilon$ is a convex combination of $p_{-2}(x^*)$, $p_0(x^*)$ and $p_2(x^*)$ for any value of X^3. We therefore assume in the following that $h < Kh_c$. The proof consists of an extensive case study, visualized in Figure 5.3.

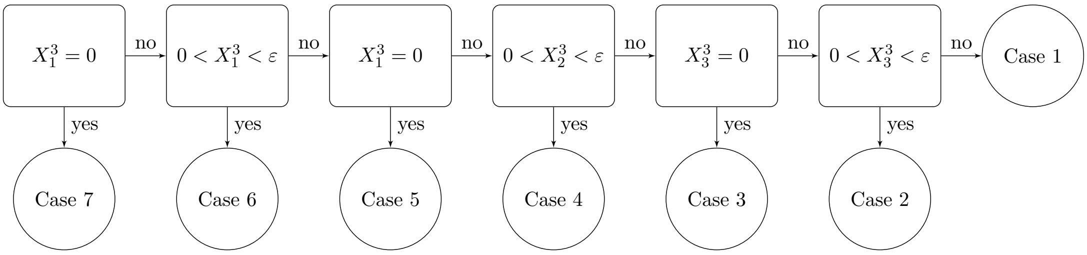


Figure 5.3: Overview of the case study done in the proof of Theorem 5.11.

Case 1: In this case $\alpha = 1$ and $\beta = 1$, therefore $\hat{f}_\varepsilon = p_{-2}(x^*) = f^*$.

Case 2: We have that $\alpha = 1$ and $0 < X_3^3 < \varepsilon$, therefore $\hat{f}_\varepsilon = (1-\beta)p_2(x^*) + \beta p_{-2}(x^*)$ where β can take any value in $(0,1)$. From (5.16), it follows that the condition $0 < X_3^3 < \varepsilon$ corresponds to

$$0 < |b_{-2} - b_2| - x^*|a_{-2} - a_2| < \varepsilon, \tag{5.22}$$

where a_{-2} and a_2 are the slopes of p_{-2} and p_2, respectively. Recall that in (5.16) the assumption that $x_i^k = i$ was made. It can however be seen that X_3^3 is invariant to grid translations and scaling. Indeed, when the grid size is scaled by h one needs to replace x^* by hx^* and $a_{\pm 2}$ by $a_{\pm 2}/h$; this leaves X_3^3 unchanged. If we now observe that from $\alpha = 1$ follows that $|a_{-2} - a_2| \neq 0$, then we can define y as the unique intersection point of p_{-2} and p_2, and obtain

$$0 < y - x^* < \frac{\varepsilon}{|a_{-2} - a_2|}. \tag{5.23}$$

Furthermore note that we can write $p_{\pm 2}(x) = a_{\pm 2}(x - y) + p_{\pm 2}(y)$ where by definition $p_{-2}(y) = p_2(y)$. This leads to the estimate

$$|p_{-2}(x^*) - p_2(x^*)| = |a_{-2} - a_2|(y - x^*) < \varepsilon. \tag{5.24}$$

Case 3: In this case $\alpha = 1$ and $\beta = 0$, therefore $\hat{f}_\varepsilon = p_2(x^*) = f^*$.

Case 4: By looking at the definition of X_2^3 in (5.16), we see that $0 < h|(a_{-2} - a_2)| < \varepsilon$ (where the factor h was added to remove the assumption that $x_i^k = i$). Furthermore we have that $X_1^3 > 0$. If $z \notin I_{-1} \cup I_0 \cup I_1$ then Lemma 5.6 and Lemma 5.8 guarantee that $\hat{f}_\varepsilon$ is a second-order accurate approximation of f^*. If $z \in I_{\pm 1}$, then p_0 is a second-order accurate approximation of $p_{\mp 2}$ on I_0. In addition, this leads to the bound

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_{\mp 2}(x^*)| + |p_{\mp 2}(x^*) - p_0(x^*)| \\ &= |(a_2 - a_{-2})(x^* - y)| + |p_{\mp 2}(x^*) - p_0(x^*)| \\ &\leqslant \frac{3}{2}\varepsilon + Ch^2 \sup|f''|. \end{aligned} \tag{5.25}$$

Finally we treat the case where $z \in I_0$. Define p_0^* as the affine function through $(x_{-1}, p_{-2}(x_{-1}))$ and $(x_0, p_2(x_0))$. It then follows from $h|(a_{-2} - a_2)| < \varepsilon$ that $h|(a_{\pm 2} - a_0^*)| < \varepsilon$ where a_0^* is the slope of p_0^*. We also have that p_0^* is a second-order accurate approximation of p_0 on the interval I_0. This then leads to

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_0^*(x^*)| + |p_0^*(x^*) - p_0(x^*)| \\ &\leqslant |(a_{\pm 2} - a_0^*)\frac{h}{2}| + |p_0^*(x^*) - p_0(x^*)| \\ &\leqslant \frac{\varepsilon}{2} + Ch^2 \sup|f''|. \end{aligned} \tag{5.26}$$

Case 5: In this case $\alpha = 0$ and therefore $\hat{f}_\varepsilon = p_0(x^*) = f^*$.

Case 6: In this case we only know that I_0 is a bad interval, since $X_1^3 > 0$. It may or may not contain the discontinuity. If $z \notin I_{-1} \cup I_0 \cup I_1$ then Lemma 5.6 and Lemma 5.8 guarantee that $\hat{f}_\varepsilon$ is a second-order accurate approximation of f^*. We therefore assume in the following that $z \in I_{-1} \cup I_0 \cup I_1$.

In the proof of Theorem 5.10, we introduced the quantity X_q^2 as a smoothness indicator for the interval I_q. We first investigate how the quantities X_q^2 for f are related to the same quantities for the piecewise linear function defined by $g(x) = f(z) + f'(z-)(x-z) + [f'](x-z)_+$, which we will denote by $\widehat{X}_q^2$. Let $\Delta_q = \Delta_h^2 f(x_q)$ and $\widehat{\Delta}_q = \Delta_h^2 g(x_q)$, as defined in (5.9), and define $\Phi_q = \Delta_q - \widehat{\Delta}_q$. Since g is a second-order accurate approximation of f, we obtain that

$$|\Phi_q| \leqslant C_q h^2 \sup|f''|, \tag{5.27}$$

where the constant C_q is independent of f and h. Using the triangle inequality and its reverse, we find that for $m, n \in \mathbb{N}_0$,

$$\begin{aligned} |\widehat{\Delta}_m| - |\widehat{\Delta}_n| &\leqslant |\widehat{\Delta}_m + \Phi_m| - |\widehat{\Delta}_n + \Phi_n| + |\Phi_m| + |\Phi_n| \\ &\leqslant |\Delta_m| - |\Delta_n| + (C_m + C_n)h^2 \sup|f''|. \end{aligned} \tag{5.28}$$

Now assume that $0 < X_q^2 < \varepsilon$ for some index q. We can then conclude that

$$0 \leqslant \widehat{X}_q^2 \leqslant X_q^2 + Ch^2 \sup|f''| < \varepsilon + Ch^2 \sup|f''|. \tag{5.29}$$

This allows us to restrict our further calculations to the piecewise linear function g. We assume that $z \in I_s$ for some index s, and that $[f'] > 0$. Furthermore we denote by $\overline{x}$ the midpoint of I_s. In Table 5.2 we calculate $\widehat{X}_{s-1}^2, \widehat{X}_s^2$ and $\widehat{X}_{s+1}^2$. Note that for $s \notin \{-1, 0, 1\}$, second-order accuracy is immediate.

First assume that $s = 0$, in this case $x^* = \overline{x}$. From the table, it follows that $\widehat{X}_0^2 \geqslant [f']h/2$. Combining this with (5.29) leads to the bound $[f']h/2 < \varepsilon + Ch^2 \sup|f''|$. We define again p_0^* as the affine function through $(x_{-1}, p_{-2}(x_{-1}))$ and $(x_0, p_2(x_0))$, which is a second-order accurate approximation of p_0. We also introduce two new second-order accurate approximations of p_0 on I_0,

$$\begin{aligned} p_0^-(x) &= p_{-2}(x) + [f']\frac{(x_0 - z)_+}{h}(x - x_{-1}), \\ p_0^+(x) &= p_2(x) + [f']\frac{(z - x_{-1})_+}{h}(x - x_0). \end{aligned} \tag{5.30}$$

x_q	x_{s-2}	x_{s-1}	x_s	x_{s+1}
$\lvert\widehat{\Delta}_q\rvert$	0	$[f'](x_s - z)$	$[f'](z - x_{s-1})$	0
$(\lvert\widehat{\Delta}_q\rvert - M_q)_+$	0	$2[f'](\overline{x} - z)_+$	$2[f'](z - \overline{x})_+$	0
$(\lvert\widehat{\Delta}_q\rvert - N_q^+)_+$	0	$2[f'](\overline{x} - z)_+$	$[f'](z - x_{s-1})$	0
$(\lvert\widehat{\Delta}_q\rvert - N_q^-)_+$	0	$[f'](x_s - z)$	$2[f'](z - \overline{x})_+$	0
$\widehat{X}_{s-1}^2, \widehat{X}_s^2, \widehat{X}_{s+1}^2$		$2[f'](\overline{x} - z)_+$	$[f'](2\lvert\overline{x} - z\rvert + \min\{x_s - z, z - x_{s-1}\})$	$2[f'](z - \overline{x})_+$

Table 5.2: Calculation of $\widehat{X}_{s-1}^2, \widehat{X}_s^2$, and $\widehat{X}_{s+1}^2$.

Indeed, $p_0^-(x_{-1}) = p_0^*(x_{-1}) = p_{-2}(x_{-1})$ and $p_0^-(x_0) = p_{-2}(x_0) + [f'](x_0 - z)_+$ are both second-order accurate on I_0, therefore p_0^- is a second-order accurate approximation on I_0 of p_0^* and hence of p_0. A similar reasoning holds for p_0^+. This then leads to

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_0^{\pm}(x^*)| + |p_0^{\pm}(x^*) - p_0(x^*)| \\ &\leqslant [f']h/2 + |p_0^{\pm}(x^*) - p_0(x^*)| \\ &\leqslant \varepsilon + Ch^2 \sup|f''|. \end{aligned} \tag{5.31}$$

Next we assume that $s = 1$. Since $X_1^3 > 0$, we have that $x_0 \leqslant z < \overline{x}$. We distinguish two subcases. First, if $x_0 \leqslant z \leqslant \overline{x}/2$ then $X_1^3 = 2[f'](\overline{x} - z)_+ \geqslant [f']h/2$. Equation (5.29) is still valid and a calculation as in (5.31) leads to the bound

$$|p_{\pm 2}(x^*) - p_0(x^*)| \leqslant \frac{5}{4}\varepsilon + Ch^2 \sup|f''| \tag{5.32}$$

where we used that $(z - x_{-1})_+ \leqslant 5h/4$. Second, if $\overline{x}/2 < z < \overline{x}$ then the intersection point y of p_{-2} and p_2 will be located in I_1. This result can be deduced from the proof of Lemma 3 in [Aràndiga et al., 2005], by taking $I = I_s$ in the beginning of the proof. The second-order accuracy then follows from Case 1 or Case 2, depending on the value of X_3^3. The case $s = -1$ is completely analogous.

Case 7: In this case $\alpha = 0$ and therefore $\hat{f}_\varepsilon = p_0(x^*) = f^*$.

We can now summarize all seven cases by the error bound

$$|\hat{f}_\varepsilon - f^*| \leqslant Ch^2 \sup|f''| + \frac{3}{2}\varepsilon, \tag{5.33}$$

which concludes the proof. □

Theorem 5.12 *Let $f : [c, d] \to [-1, 1]$ be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. For every $\varepsilon > 0$, there exists a pure ReLU neural network with input $f^k \in [-1, 1]^{N_k+1}$ and output $\hat{f}_{2i-1,\varepsilon}^{k+1}$ for every $1 \leqslant i \leqslant N_k$, $0 \leqslant k \leqslant K$.*

Proof Most of the work was already done in Theorem 5.5 and the discussion preceding Theorem 5.11. Indeed, we have already established that $p_{i-2}^k(x_{2i-1}^{k+1}), p_i^k(x_{2i-1}^{k+1})$, $p_{i+2}^k(x_{2i-1}^{k+1}), X_1^3, X_2^3$ and X_3^3, as well as the operation $\star$ can be represented using pure ReLU networks. It only remains to find a bound for all second arguments of the operation $\star$ in (5.21). Since the codomain of f is $[-1, 1]$, one can calculate that

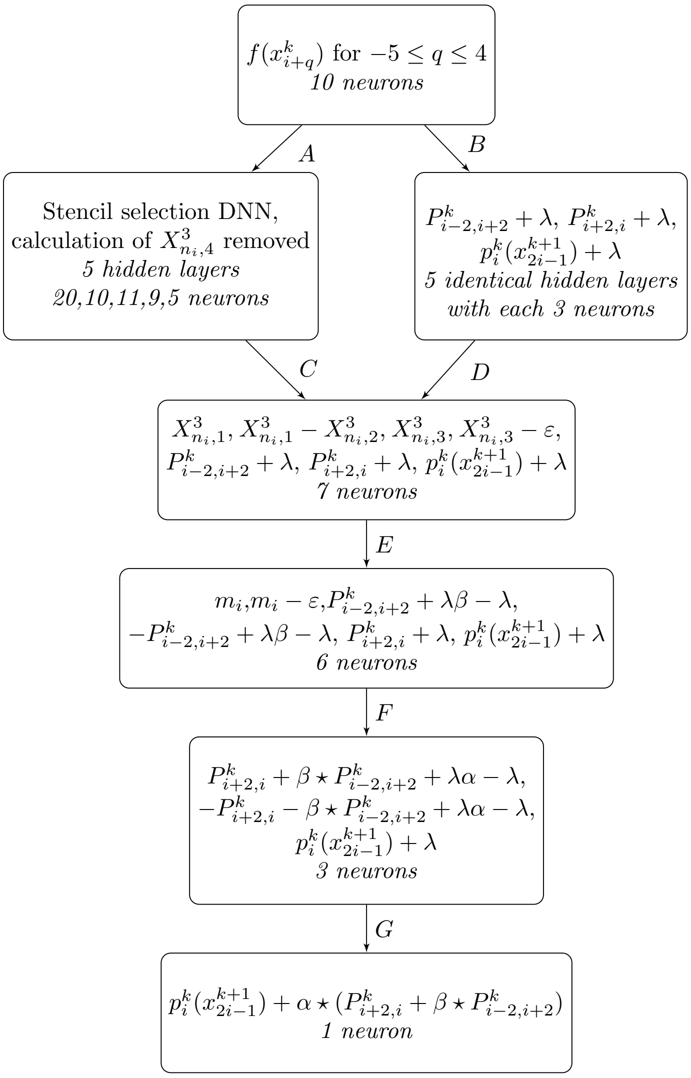

$f(x^k_{i+q})$ for $-5 \leq q \leq 4$
10 neurons
A
B
Stencil selection DNN,
calculation of $X^3_{n_i,4}$ removed
5 hidden layers
20,10,11,9,5 neurons
$P^k_{i-2,i+2} + \lambda$, $P^k_{i+2,i} + \lambda$,
$p^k_i(x^{k+1}_{2i-1}) + \lambda$
5 identical hidden layers
with each 3 neurons
C
D
$X^3_{n_i,1}$, $X^3_{n_i,1} - X^3_{n_i,2}$, $X^3_{n_i,3}$, $X^3_{n_i,3} - \varepsilon$,
$P^k_{i-2,i+2} + \lambda$, $P^k_{i+2,i} + \lambda$, $p^k_i(x^{k+1}_{2i-1}) + \lambda$
7 neurons
E
m_i,$m_i - \varepsilon$,$P^k_{i-2,i+2} + \lambda\beta - \lambda$,
$-P^k_{i-2,i+2} + \lambda\beta - \lambda$, $P^k_{i+2,i} + \lambda$, $p^k_i(x^{k+1}_{2i-1}) + \lambda$
6 neurons
F
$P^k_{i+2,i} + \beta \star P^k_{i-2,i+2} + \lambda\alpha - \lambda$,
$-P^k_{i+2,i} - \beta \star P^k_{i-2,i+2} + \lambda\alpha - \lambda$,
$p^k_i(x^{k+1}_{2i-1}) + \lambda$
3 neurons
G
$p^k_i(x^{k+1}_{2i-1}) + \alpha \star (P^k_{i+2,i} + \beta \star P^k_{i-2,i+2})$
1 neuron

Figure 5.4: Flowchart of a ReLU DNN to calculate $\hat{f}^{k+1}_{2i-1,\varepsilon}$ from f^k.

$p^k_{i-2}(x^{k+1}_{2i-1}), p^k_i(x^{k+1}_{2i-1})$ and $p^k_{i+2}(x^{k+1}_{2i-1})$ lie in $[-4, 4]$. Using Lemma 4.7, we then find that

$$p^k_{i+2}(x^{k+1}_{2i-1}) - p^k_i(x^{k+1}_{2i-1}) + \beta \star \left(p^k_{i-2}(x^{k+1}_{2i-1}) - p^k_{i+2}(x^{k+1}_{2i-1})\right) \in [-16, 16]$$

for all $\beta \in [0, 1]$. We can thus use the operation $\star$ with $\lambda = 16$ in Definition 4.6. □

We now present the network architecture of a ReLU neural network that computes the approximate ENO-SR prediction (5.21). The network we propose is visualized in Figure 5.4 and consists of eight hidden layers with widths 23, 13, 14, 12, 8, 7, 6 and 3. In the figure, the following notation was used,

$$\begin{aligned} m_i &= \min\{X^3_{n_i,1}, X^3_{n_i,2}\}, \\ P^k_{m,n} &= p^k_m(x^{k+1}_{2i-1}) - p^k_n(x^{k+1}_{2i-1}), \end{aligned} \tag{5.34}$$

for $1 \leqslant i, m, n \leqslant N_k$. We now give some more explanation about how all the layers can be calculated from the previous layer in Figure 5.4. ***A.B.*** All quantities of the first layer are linear combinations of the input neurons, where we also refer to Figure 5.2. From the proof of Theorem 5.12, it follows that we can take $\lambda = 16$. ***C.D.*** Linear combinations. ***E.*** We refer to (5.21) and (5.34) for the definitions of β and m_i, respectively. ***F.*** We refer to Definition 4.6 and (5.21) for the definitions of $\star$ and α, respectively. ***G.*** From (5.21) it follows that the value of the output layer is indeed equal to the approximate second-order ENO-SR prediction $\hat{f}^{k+1}_{2i-1,\varepsilon}$.

Theorem 5.12 can be used in two different ways. First, it justifies to train a network with the architecture as in Figure 5.4, with data provided by the approximate second-order ENO-SR algorithm. In practice, we can just use the exact adapted ENO-SR algorithm, since we can take $\varepsilon > 0$ arbitrarily small. Second, this theorem proves that there exists a pure ReLU network whose output satisfies the bound from Theorem 5.11. Using that ENO-SR-2 is second-accurate itself, we can use any 'approximately second-order accurate algorithm' to produce the training data, including the actual function values on the finer grid. This approach will however not be pursued any further in this thesis.

Chapter 6

Numerical results

From the two previous chapters, we know that there exist deep ReLU neural networks, of a certain architecture that will mimic the ENO-p and the second-order ENO-SR algorithms for interpolating rough functions. Next, we proceed to recreate or *train* these networks based on a finite data set and investigate their performance. From now on, we will refer to these networks as *DeLENO* (Deep Learning ENO) and *DeLENO-SR* networks. More details on the training procedure can be found in Section 6.1, the performance is discussed in Section 6.2 and illustrated by various applications at the end of the chapter.

6.1 Training procedure

The *training* of these networks involves finding a parameter θ (the weights and biases of the network) that approximately minimizes a certain *loss function* $\mathcal{J}$ that measures the error in the network's predictions. To achieve this, we have access to a finite data set $\mathbb{S} = \{(X^i, f(X^i))\}_i \subset D \times f(D)$, where $f : D \subset \mathbb{R}^m \to \mathbb{R}^n$ is the unknown function we try to approximate using a neural network f^θ.

For classification problems, each $Y^i = f(X^i)$ is an n-tuple that indicates to which of the n classes X^i belongs. The output of the network $\hat{Y}^i = f^\theta(X^i)$ is an approximation of Y^i in the sense that $\hat{Y}^i_j$ can be interpreted as the probability that X^i belongs to class j. A suitable loss function in this setting is the *cross-entropy function* with regularization term

$$\mathcal{J}(\theta; \mathbb{S}, \lambda) = -\frac{1}{\#\mathbb{S}} \sum_{(X^i,Y^i)\in\mathbb{S}} \sum_{j=1}^{n} Y^i_j \log(\hat{Y}^i_j) + \lambda\mathcal{R}(\theta). \tag{6.1}$$

The cross-entropy term measures the discrepancy between the probability distributions of the true outputs and the predictions. It is common to add a regularization term $\lambda\mathcal{R}(\theta)$ to prevent overfitting of the data and thus improve the generalization capabilities of the network [Goodfellow et al., 2016]. The network hyperparameter $\lambda > 0$ controls the extent of regularization. Popular choices of $\mathcal{R}(\theta)$ include the sum of some norm of all the weights of the network. It is advisable to not regularize the network biases as it can lead to underfitting the data [Goodfellow et al., 2016]. To monitor the generalization capability of the network, it is useful to split $\mathbb{S}$ into a training set $\mathbb{T}$ and a validation set $\mathbb{V}$ and minimize $\mathcal{J}(\theta; \mathbb{T}, \lambda)$ instead of $\mathcal{J}(\theta; \mathbb{S}, \lambda)$. The validation set $\mathbb{V}$ is used to evaluate the generalization error. The accuracy of network f^θ on $\mathbb{T}$ is measured as

$$\mathbb{T}_{acc} = \# \left\{ (X,Y) \in \mathbb{T} \mid \hat{Y} = f^\theta(X), \quad \arg\max_{1 \leqslant j \leqslant n} \hat{Y}_j = \arg\max_{1 \leqslant j \leqslant n} Y_j \right\} / \# \mathbb{T}, \tag{6.2}$$

with a similar expression for $\mathbb{V}_{acc}$.

For regression problems, $\hat{Y}^i$ is a direct approximation of Y^i, making the *mean squared error* with regularization term

$$\mathcal{J}(\theta; \mathbb{S}, \lambda) = \frac{1}{\#\mathbb{S}} \sum_{(X^i,Y^i)\in\mathbb{S}} \|Y^i - \hat{Y}^i\|^2 + \lambda \mathcal{R}(\theta), \tag{6.3}$$

an appropriate loss function. Analogously to the classification setting, the data set $\mathbb{S}$ can be split into a training set $\mathbb{T}$ and a validation set $\mathbb{V}$, in order to minimize $\mathcal{J}(\theta; \mathbb{T}, \lambda)$ and estimate the MSE of the trained network by $\mathcal{J}(\theta; \mathbb{V}, \lambda)$.

In the previous chapters, we have proven the existence of ReLU neural networks that approximate ENO(-SR) well, or can even exactly reproduce its output. It remains however a great challenge to reobtain these networks based on a finite training set. In particular, the loss function $\mathcal{J}$ is usually highly non-convex, giving rise to a large number of local minima. The minimization of $\mathcal{J}$ is therefore generally performed using a stochastic iterative gradient algorithm. Such an algorithm splits the training data set into a number of mini-batches, followed by taking an optimization step over every mini-batch. A complete pass through the whole data set is called an *epoch*. After each epoch, the data set is reshuffled and new mini-batches are created. It is this stochasticity that assists the training algorithm to escape local minima. A single (generic) optimization step is of the form

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{J}(\theta_t; \mathbb{B}_k, \lambda), \tag{6.4}$$

where η_t is the learning rate and $\mathbb{B}_k$ is a mini-batch of the data set. The learning rate is an important hyperparameter that controls the step size while moving in the direction of steepest descent. It is usually taken as a decreasing function of t. In this thesis we used for instance

$$\eta_t = \eta_{t-1} \frac{1}{1+\beta t}, \quad \eta_0 = 1.0 \cdot 10^{-3}, \quad \beta = 1.0 \cdot 10^{-5} \tag{6.5}$$

and a mini-batch size of 1024. Several other, more advanced, stochastic optimizers have been developed, among which a popular algorithm is the ADAM optimizer [Kingma and Ba, 2014]. When the simple optimization algorithm given by (6.4) and (6.5) did not provide satisfying results, we used the ADAM optimizer, again with a mini-batch size of 1024. More information on the training of the DeLENO(-SR) classification and regression networks can be found in the following subsections.

6.1.1 Training data sets

Apart from using a suitable optimization algorithm, the quality of the training data set is of the utmost importance. In the previous chapters, we proved the existence of networks that either exactly agree or approximate the ENO and ENO-SR interpolation procedures. An unrepresentative training data set can cause the trained network to significantly deviate from the interpolation procedure it is supposed to agree with. In what follows, we list the different types of data we will use during the training. We assume that the input of the network is m-dimensional.

- *Random samples.* We generate samples $X \in \mathbb{R}^m$, with each component X_j randomly drawn from the uniform distribution on the interval $[-1, 1]$.
- *Sinusoidal samples* of the form

  $$\{(u_l, ..., u_{l+m})^\top \mid 0 \leqslant l \leqslant N-m, \quad 0 \leqslant q \leqslant 39\}$$

  where u_l is defined as

  $$u_l := \sin\left((q+1)\pi \frac{l}{N}\right), \quad 0 \leqslant l \leqslant N, \quad N \in \mathbb{N}.$$

- *Piecewise smooth samples.* We use the piecewise version of n-term Taylor polynomials p_n defined by

  $$p_n(x) = \sum_{k=0}^{n} (a_k(x-z)_-^k + b_k(x-z)_+^k)/k!, \tag{6.6}$$

  for $x \in \mathbb{R}$ with parameters $z, a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{R}$ and evaluate the function on an m-dimensional stencil.

Furthermore, the input data needs to be appropriately scaled before being fed into the network, to ensure faster convergence during training and to improve the networks ability to generalize. The following two scaling functions can be used to scale the input X,

$$\text{Scale}_1(X) = \begin{cases} \frac{2X-(\max_j X_j + \min_j X_j)}{\max_j X_j - \min_j X_j} & \text{if } X \neq 0 \\ (1, \ldots, 1)^\top \in \mathbb{R}^m & \text{otherwise} \end{cases}, \tag{6.7a}$$

$$\text{Scale}_2(X) = \begin{cases} \frac{2X-(\max_j X_j + \min_j X_j)}{\max_j X_j - \min_j X_j} & \text{if } \max_j X_j - \min_j X_j > 2 \\ X & \text{otherwise} \end{cases}, \tag{6.7b}$$

which both scale the input to lie in the box $[-1, 1]^m$.

Remark 6.1 *When the input data is scaled using formula* (6.7a)*, then Newton's undivided differences are scaled by a factor* $2(b-a)^{-1}$ *as well. Therefore scaling does not alter the stencil shift obtained using Algorithm 1. The same holds for scaling using formula* (6.7b)*.*

6.1.2 Training details

Next, we build a suitable training set for each network type, using the different sample types from the previous subsection. In addition, we list other training details such as the the value that was used for the regularization parameter.

DeLENO-p classification. We want to construct a suitable training data set $\mathbb{S}$ to train a DeLENO-p classification network for interpolation purposes. The network will take an input from $\mathbb{R}^m$, with $m = 2p-2$, and predict the stencil shift r. We generate a data set $\mathbb{S}$ of size 460200-200m using Algorithm 1 with inputs given by:

- 400000 m-dimensional random samples.

- 60200-200m sinusoidal samples obtained by setting $N = 100, 200, 300, 400, 500$.

Thanks to the results of Section 4.3, we are guaranteed that for certain architectures it is theoretically possible to achieve an accuracy of 100%. For any order, this architecture is given by Theorem 4.2 and its proof. For small orders $p = 3, 4$ we take the alternative network architectures described at the end of Section 4.3, as they are smaller. The loss function $\mathscr{J}$ is chosen as (6.1), with an L_2 penalization of the network weights and $\lambda = 7.8 \cdot 10^{-6}$. The network is retrained using 5 times for 2000 epochs each, with the weights and biases initialized using a random normal distribution. The last 20% of $\mathbb{S}$ is set aside to be used as the validation set $\mathbb{V}$. For each p, we denote by DeLENO-p the network with the highest accuracy $\mathbb{V}_{acc}$ at the end of the training. The accuracies achieved for DeLENO interpolation networks are given in Table 6.1. The training of the DeLENO reconstruction networks was performed entirely analogously, with the only difference that now $m = 2p - 1$.

DeLENO-p regression. In the case of DeLENO-p regression smooth sample functions are required, therefore we take $a_k = b_k$ for $1 \leqslant k \leqslant n$ and we generate them from the uniform distribution on $[-1, 1]$. Furthermore we set $z = 0$ and assume that the x-values of the stencil are $-1, -\frac{p-2}{p} \ldots, 1$. Since we will only train networks for $p = 3, 4$, we take $n = 4$. Unless stated otherwise, we use a training data set of 500000 piecewise smooth samples, which can optionally be extended by 50000 random samples. In this way, the training set is hopefully sufficiently rich to achieve p-th-order accuracy using the trained networks. Unless stated otherwise, the network architectures of the networks are those of Section 4.4. Other training details are the same as for the DeLENO classification networks.

DeLENO-SR classification and regression. Recall that ENO-SR is designed to interpolate continuous functions f that are two times differentiable, except at a single point $z \in \mathbb{R}$ where the first derivative has a jump of size $[f']$. Locally, these functions look like piecewise linear functions. Based on this observation, we create a data set using $n = 1$. Higher values of n theoretically lead to a richer data set (i.e. one that approximates the function class of interest better), but this was not reflected in the training results. For notational simplicity we assume that the x-values of the stencil that serves as input for the ENO-SR algorithm (Section 5.2) are $0, 1, \ldots, 9$. The interval of interest is then $[4, 5]$ and the goal of ENO-SR is to find an approximation of f at $x = 4.5$. We generate 100000 samples, where we choose a_1, b_1, z in the following way:

- The parameters a_1 and b_1 are drawn from the uniform distribution on the interval $[-1, 1]$. Note that any interval that is symmetric around 0 could have been used, since the data will be scaled afterwards.
- For 25000 samples, z is drawn from the uniform distribution on the interval $[4, 5]$. This simulates the case where the discontinuity is inside the interval of interest.
- For 75000 samples, z is drawn from the uniform distribution on the interval $[-9, 9]$, which also includes the case in which f is smooth on the stencil.

The network architectures are described in Section 5.2. The networks will take an input from $\mathbb{R}^{10}$ and either predict the stencil shift r or directly give an approximation of the output of the ENO-SR algorithm. The training of DeLENO-SR was performed in a very similar fashion to the training of DeLENO-p, only this time we retrained the DeLENO-SR network 5 times for 5000 epochs each. Furthermore we used 8-fold cross-validation on a data set of 20000 samples to select the optimal regularization parameter, resulting in the choice $\lambda = 1 \cdot 10^{-8}$.

Remark 6.2 *Note that the detection mechanism of the ENO-SR interpolation method (Section 5.2) labels an interval as bad when $\alpha - \beta > 0$ for some numbers $\alpha, \beta \in \mathbb{R}$. This approach causes poor approximations in practice due to numerical errors. When for example $\alpha = \beta$, rounding can have as a consequence that* $\mathrm{round}(\alpha - \beta) > 0$*, leading to an incorrect label. This deteriorates the accuracy of the method and is very problematic for the training. Therefore we used in our code the alternative detection criterion $\alpha - \beta > \varepsilon$, where for example $\varepsilon = 10^{-10}$.*

6.2 Performance of DeLENO(-SR) methods

We discuss the performance of the trained DeLENO(-SR) networks based on four test functions $q_1, q_2, q_3, q_4 : [0,1] \to \mathbb{R}$ which are defined by

$$q_1(x) = -\left(x - \frac{\pi}{6}\right) \cdot \mathbb{1}_{(-\infty, \frac{\pi}{6}]}(x) + \left(x - \frac{\pi}{6}\right)^2 \cdot \mathbb{1}_{(\frac{\pi}{6}, +\infty)}(x), \tag{6.8}$$

$$q_2(x) = \sin(x/10), \tag{6.9}$$

$$q_3(x) = \sin(x), \tag{6.10}$$

$$q_4(x) = \sin(10x). \tag{6.11}$$

These specific functions were chosen as they illustrate the performance of the different networks well for functions of distinctive regularity. However, it was observed that for some particular functions, the approximation accuracy reduced considerably compared to the general behaviour as described in the following subsections. We assess the quality of the approximation by their accuracy (6.2) for classification networks, by the MSE (6.3) for regression networks and by the order of accuracy for both.

6.2.1 DeLENO classification networks

The accuracies for DeLENO classification networks for both interpolation and reconstruction purposes are listed in Table 6.1. Scaling function Scale$_1$ was used. It can be seen that DeLENO and ENO agree almost perfectly on the training data set, as well as the part of the data set that was left out during the training process.

type	p	hidden layer sizes	$\mathbb{T}_{acc}$	$\mathbb{V}_{acc}$
interpolation	3	4	99.36%	99.32%
interpolation	4	10,6,4	99.22%	99.14%
reconstruction	2	4	99.96%	99.97%
reconstruction	3	10,6,4	99.65%	99.65%

Table 6.1: Shape of DeLENO-p and DeLENO-SR networks with their accuracies for the interpolation and reconstruction problem.

The obtained weights and biases for the trained third- and fourth-order DeLENO interpolation network are listed in Appendix A. As the networks do not have an accuracy of 100%, it comes as no surprise that the listed matrices and biases differ significantly from their theoretical counterparts, which can be found in equations (4.19-4.24). This shows that there are multiple neural networks which can approximate the ENO interpolation procedure very well.

We investigate the effect of the minor disagreement between ENO and DeLENO on the interpolation accuracy. With respect to the L^∞ norm, this small discrepancy might have a significant impact. Figure 6.1 however shows that this is not the case: for the smooth function q_4 and the piecewise smooth function q_1, the DeLENO-p classification networks lead to exactly the same result as ENO-p for $p = 3, 4$. As the error is evaluated on a different grid for every grid size, the course of the graph deviates from a straight line, as can be seen for q_1 in Figure 6.1 (right).

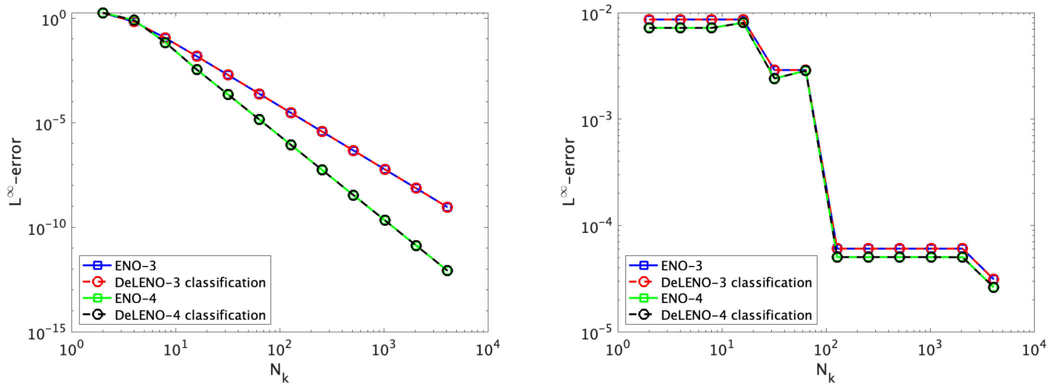


Figure 6.1: Interpolation error when using DeLENO classification networks for the smooth function q_4 (left) and the piecewise smooth function q_1 (right).

6.2.2 DeLENO regression networks

We first discuss the performance of the third-order DeLENO regression networks when scaling function Scale_1 is used. The network was trained on a training set of 100000 samples consisting of fourth-order Taylor expansions. Best results were seen when the regularization parameter was taken to be $\lambda = 10^{-8}$.

We discuss the performance of the network based on the test functions q_1, q_2, q_3, q_4. We adopt the setting of Section 4.1 and set $N_0 = 16$ and $K = 11$. In particular, we approximate f^{k+1} using the ENO-3 interpolation procedure (cf. (4.13)) and a DeLENO-3 regression network, both times with f^k as input data. We denote these approximations by $\text{ENO}_3[f^k]$ and $\text{DeLENO}_3^1[f^k]$, respectively, where the superscript 1 refers to Scale_1. In Figure 6.2, we plot $\|\text{ENO}_3[f^k] - f^{k+1}\|_\infty$ in blue, $\|\text{DeLENO}_3^1[f^k] - f^{k+1}\|_\infty$ in red and $\|\text{ENO}_3[f^k] - \text{DeLENO}_3^1[f^k]\|_\infty$ in black. For the piecewise smooth function q_1, ENO-3 and DeLENO-3 visually agree and are first-order accurate, which is indeed the expected order of accuracy. Interestingly, the discrepancy between ENO-3 and DeLENO-3 is linearly proportionate to the grid size. This behaviour is also observed for q_2, q_3 and q_4. ENO-3 is third-order accurate, but DeLENO-3 is only first-order accurate for the smooth functions q_2 and q_3. For q_4, ENO-3 and DeLENO-3 agree on coarse grids, but beyond a critical grid size, the third-order accuracy reduces to first-order accuracy. We note that everywhere where DeLENO-3 is only first-order accurate,

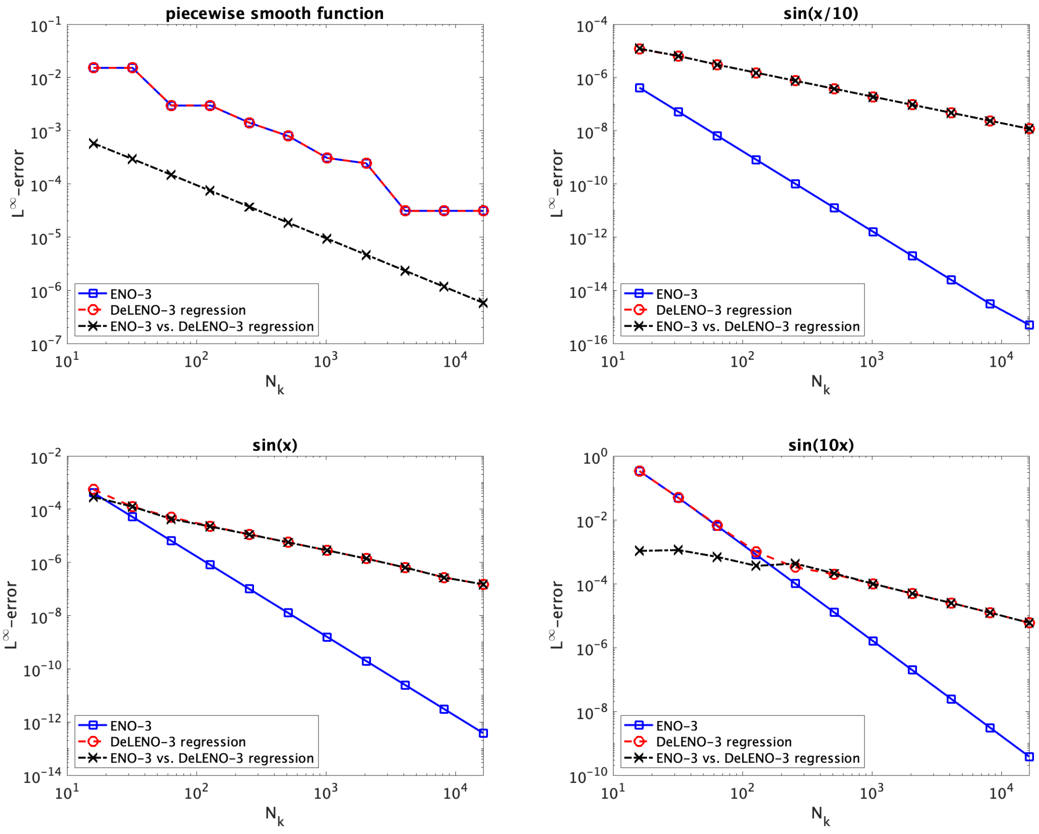


Figure 6.2: Plot of $\|\mathrm{ENO}_3[f^k] - f^{k+1}\|_\infty$, $\|\mathrm{DeLENO}_3^1[f^k] - f^{k+1}\|_\infty$ and $\|\mathrm{ENO}_3[f^k] - \mathrm{DeLENO}_3^1[f^k]\|_\infty$ as a function of N_k for the test functions q_1, q_2, q_3 and q_4.

its approximation error agrees with the discrepancy between ENO-3 and DeLENO-3. These observations can be summarized as

$$\|\mathrm{DeLENO}_3^1[f^k] - f^{k+1}\|_\infty \approx \|\mathrm{DeLENO}_3^1[f^k] - \mathrm{ENO}_3[f^k]\|_\infty + \|\mathrm{ENO}_3[f^k] - f^{k+1}\|_\infty. \tag{6.12}$$

In other words, the DeLENO approximation error can be decomposed in how well ENO approximates the test function on the one hand, and how well DeLENO approximates ENO on the other hand. This observation also allows an explanation of the orders of accuracy as observed in Figure 6.2.

First, we note that there are no outliers in the errors of the predictions of the DeLENO-3 regression network, which makes the MSE (6.3) fairly robust to the size of the data set (Figure 6.3). Keeping this in mind, we make the crude simplification that the network makes for every new prediction a constant error of the size of the square root of the MSE. In this case $\sqrt{\mathrm{MSE}} \approx 3 \cdot 10^{-3}$. However, before the network is used to predict f^{k+1}, the function values of the local stencil are rescaled to the interval $[-1, 1]$ using scaling function Scale_1 (6.7a). For smooth functions, this corresponds to the multiplication with a factor linearly proportionate to N_k. Indeed, a smooth function f varies at most $3h_k \sup_{x\in[0,1]}|f'|$ inside the local stencil of the ENO-3 method. On this scaled data, the network makes an error of size $\sqrt{\mathrm{MSE}}$, resulting in an error of size $C_1 h_k \sqrt{\mathrm{MSE}} \sup_{x\in[0,1]}|f'|$ for some constant $C_1 > 0$ on the unscaled data. We therefore

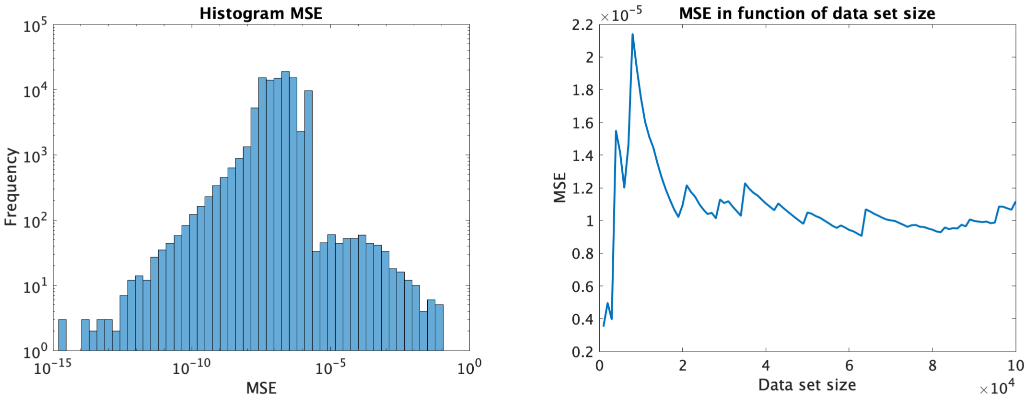


Figure 6.3: Histogram of the squared errors of the predictions of the DeLENO-3 regression network on the training data set (left) and the MSE as a function of the data set size (right).

have that

$$\|\mathrm{DeLENO}_3^1[f^k] - \mathrm{ENO}3[f^k]\|_\infty \approx C_1 h_k \sqrt{\mathrm{MSE}} \sup_{x\in[0,1]} |f'|. \tag{6.13}$$

Since in addition ENO-3 is third-order accurate, we can rewrite (6.12) as

$$\|\mathrm{DeLENO}_3^1[f^k] - f^{k+1}\|_\infty \approx C_1 h_k \sqrt{\mathrm{MSE}} \sup_{x\in[0,1]} |f'| + C_2 h_k^3 \sup_{x\in[0,1]} |f'''| \tag{6.14}$$

for some constant $C_2 > 0$. This formula explains many aspects of the phenomena observed in Figure 6.2. For the sine function q_2, the constants C_1 and C_2 of (6.14) relate in such a way that the linear term dominates over the higher-order term. Every time the frequency of the test sine function is multiplied by 10, the factor $\sup_{x\in[0,1]}|f'|$ in (6.14) increases by the same factor. This can be observed in Figure 6.2 by comparing the second, third and fourth plot, and noting that the increase of the black line matches the increase of the frequency. However, at the same time, $\sup_{x\in[0,1]}|f'''|$ increases by a factor of 1000 if the frequency is multiplied by 10, leading to a much higher increase of the blue line in the plots. Therefore the second term in (6.14) will dominate the first term for highly oscillatory functions as q_4. Finally, note that the value of $\sqrt{\mathrm{MSE}}$ is also in agreement with Figure 6.2 under the hypothesis that (6.14) holds.

In an attempt to improve the order of accuracy of the regression network, we change the scaling procedure function as the deterioration to first-order accuracy was caused by the scaling. We investigate the approximation accuracy when Scale_2, as defined in (6.7b), is used. Instead of scaling the stencil such that the maximum and minimum of the stencil are respectively 1 and -1, Scale_2 only translates and rescales the stencil such that the maximum is smaller than 1 and the minimum is larger than -1. The network obtained using the training procedure from Section 6.1 and a data set of 100000 fourth-order Taylor polynomials did not lead to a good approximation of ENO-3. There was for both smooth and piecewise smooth functions no agreement at all between the ENO-3 and the DeLENO-3 regression network using Scale_2, nor did it show the expected order of accuracy for smooth functions.

The following approaches were pursued to improve the performance of the network.

- Increase of the size of the training data set from 100000 to 500000. We refer to the obtained network as *DeLENO-3 regression A* or DeLENO$^2_{3A}$, where the superscript 2 refers to Scale$_2$. The results of this approach are shown in Figure 6.4, where the approximation errors for the piecewise smooth function q_1 (left) and a sinusoid q_3 (right) are plotted. For the smooth function, the network is third-order accurate until a certain error is reached (in this case of the order 10^{-8}) after which the error remains constant. Moreover, the network and ENO-3 make the same approximation error for q_1. Interestingly, these good results do not mean that the network and ENO-3 agree. Using the notation from the previous subsection, Figure 6.4 shows that $\|\text{DeLENO}^2_{3A}[f^k] - \text{ENO}_3[f^k]\|_\infty$ is of third order until 10^{-8} is reached. This means that we did not obtain ENO-3, but merely a third-order accurate approximation of ENO-3, at least for this test function. Moreover, the network may have not inherited all desirable properties that characterize ENO-3. This is indeed the case: the network is seen to create oscillations when interpolating functions with jump discontinuities. This is very undesirable as the total variation boundedness is one of the key properties of ENO.

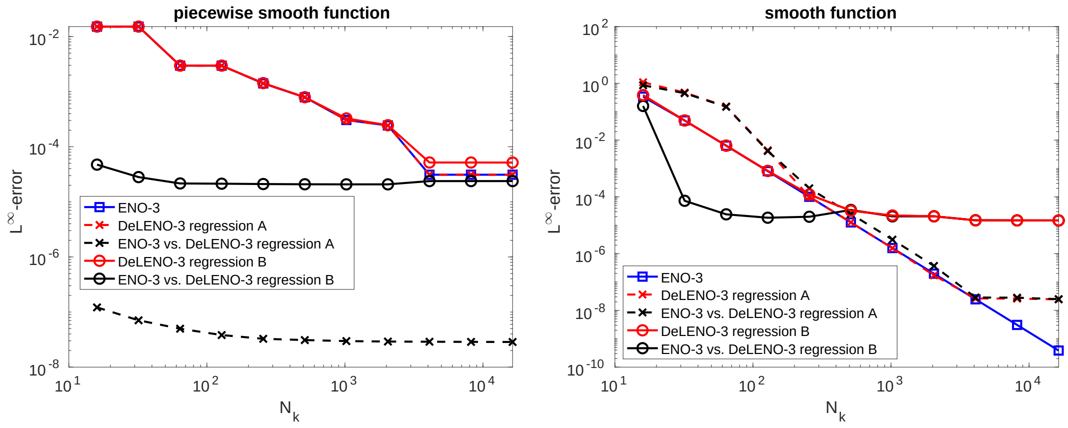


Figure 6.4: Plots of the approximation error when using the DeLENO-3 regression network for the piecewise smooth function q_1 (left) and the sine function q_4 (right). The approximation error of ENO-3 is shown for comparison. When not visible, the red crosses coincide with the black crosses.

- Increase of the size of the network, from three layers with respectively 6,4 and 3 neurons to six layers with each 10 neurons. As more data is needed to train a larger network, we increase simultaneously the data set size from 100000 to 500000. Only unmeaningful approximations were obtained using this approach.

- Use of a different loss function. For a vector $X \in \mathbb{R}^n$, $n \in \mathbb{N}$, introduce the notation $\rho(X) = \max_j X_j - \min_j X_j$. When the (regularized) MSE (6.3) is used as loss function, the loss is dominated by the errors corresponding to stencil vectors X for which $\rho(X)$ is relatively large. However, it is crucial for a good performance on fine grids that the error for stencil vectors with a small $\rho(X)$ also contribute to the loss function. We therefore introduce a family of weighted (regularized) MSE loss functions, parametrized by $k \in \mathbb{N}$,

$$\mathrm{MSE}_k(\theta; \mathbb{S}, \lambda) = \frac{1}{\#\mathbb{S}} \sum_{(X^i, Y^i) \in \mathbb{S}} \frac{\|Y^i - \hat{Y}^i\|^2}{\rho(X^i)^{2k}} + \lambda \mathcal{R}(\theta). \tag{6.15}$$

The choice $k = 1$ corresponds to, up to a constant, to the mean squared *relative* error. When trying to reobtain ENO-p using a neural network, the choice $k = p$ is relevant as well. Recall that on fine grids DeLENO$^2_{3A}$ had a constant error, this modified loss function might remove this issue. It comes however as no surprise that the weighted MSE loss for $k > 0$ is not robust towards stencils for which $\rho(X^i)$ is very small, which hampered the training. As a consequence, no meaningful networks were obtained.

- Change the composition of the data set. Recall that the network DeLENO$^2_{3A}$ was third-order accurate for smooth functions, but did not perform well on discontinuous functions. As DeLENO$^2_{3A}$ was only trained on smooth samples, this does not need to be a surprise. Therefore we add to the training set a number of random samples. The trained networks were seen to be very sensitive to the ratio between the amounts of smooth and random samples. Too little random samples lead to networks similar to DeLENO$^2_{3A}$, whereas too much random samples result in a reduction of the order of accuracy for smooth functions. In Figure 6.4, the approximation error of *DeLENO-3 regression B* (DeLENO$^2_{3B}$) is shown, a network that was obtained using a training data set of 500000 sinusoidal and 50000 random samples. The approximation error behaves similarly to that of DeLENO$^2_{3B}$, albeit resulting in a larger error for fine grids, but with the major difference that $\|\mathrm{DeLENO}^2_{3B}[f^k] - \mathrm{ENO}_3[f^k]\|_\infty$ is now a constant. This is in consonance with our hypothesis on the error decomposition of well-trained DeLENO networks (6.12) of the previous subsection. Moreover, the different behaviour on small grids between DeLENO$^1_3$ and DeLENO$^2_{3B}$ is fully in agreement with the distinct scaling procedures.

The performance of the DeLENO-4 networks are very similar to that of the DeLENO-3 networks and will not be further discussed.

6.2.3 DeLENO-SR classification network

As with the DeLENO classification networks, the accuracy of the second-order accurate DeLENO-SR classification network is very high on the training and validation data set (resp. 99.74% and 99.81%). In Figure 6.5, we check the order of accuracy for the piecewise smooth function q_1 and the smooth function q_3. Best results were obtained when using scaling function Scale$_1$. On most grids, ENO-SR-2 and the method using the DeLENO-SR-2 classification network perfectly agree. However, for very fine grids, some instabilities might arise due to the nature and implementation of the ENO-SR algorithm (cfr. Remark 6.2). This can indeed be observed for the piecewise smooth function q_1 (left graph of Figure 6.5). We also note that for coarse grids, (DeL)ENO-3 and (DeL)ENO-SR-2 make a very similar approximation error. In this case, it is thus advisable to use DeLENO instead of DeLENO-SR as the first is computationally much more efficient.

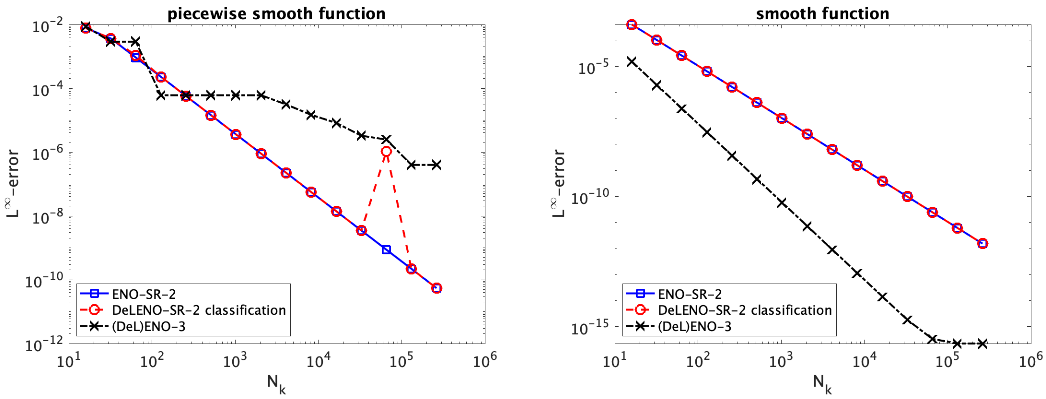


Figure 6.5: Plots of the approximation error when using the DeLENO-SR-2 classification network for the piecewise smooth function q_1 and the sine function q_3. The approximation error of ENO-SR-2 and (DeL)ENO-3 are shown for comparison.

6.2.4 DeLENO-SR regression network

The DeLENO-SR regression network was trained on a data set consisting of piecewise linear functions, as described in Section 6.1.2, and using scaling function Scale$_1$. On the validation part of the data set, it was seen that MSE $= 5 \cdot 10^{-10}$, which is considerably lower than the validation MSE of the DeLENO regression network. We therefore expect that, when using scaling function Scale$_1$, the same reduction to first-order accuracy will be observed as with DeLENO regression, but now only for very fine grids. This hypothesis is confirmed in Figure 6.6, where the approximation errors of the piecewise smooth function q_1 (left) and the sine function q_3 (right) are shown. In both cases the order of accuracy only deteriorates when $N_k \geqslant 10^4$. Although the DeLENO-SR regression network is initially second-order accurate for the smooth function, the approximation error does not agree with that of ENO-SR. This is in line with Theorem 5.11, where we proved that there exists a network that is a second-order accurate approximation of ENO-SR-2 (except for an error term that can be arbitrarily small, yet is fixed). A

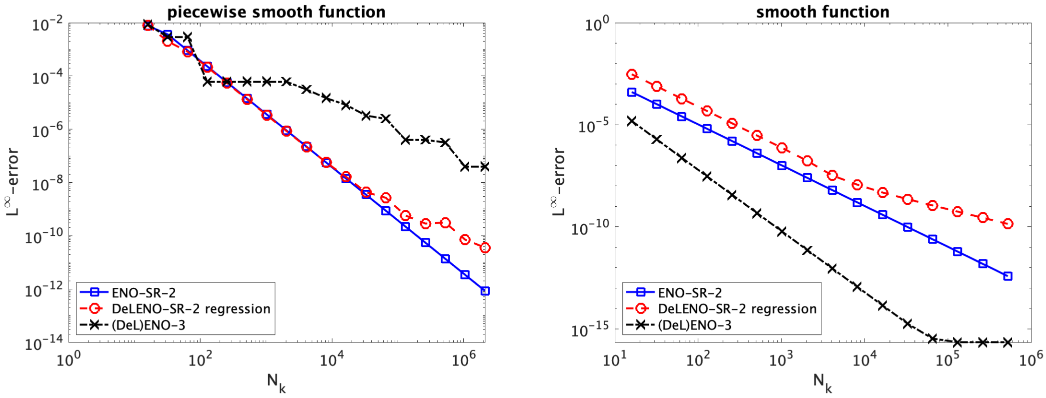


Figure 6.6: Plots of the approximation error of the DeLENO-SR-2 regression network for the piecewise smooth function q_1 and the sine function q_3. The approximation errors of ENO-SR-2 and (DeL)ENO-3 are shown for comparison.

second factor that might contribute is the fact that DeLENO-SR-2 was trained on piecewise linear functions, which can be thought of as a second-order accurate approximation of a smooth function, therefore leading to a higher error. Theoretically this error can be decreased by generating a richer data set, with for instance piecewise quadratic or cubic functions. In practice, however, a significant increase of the MSE on the training set is observed, leading to a poor overall performance as the network will only be second-order accurate for coarser grids. The same phenomenon occurred when Scale$_2$ was used as scaling function.

6.3 Applications

6.3.1 Function approximation

We demonstrate the approximating ability of the DeLENO interpolation method using the function

$$u(x) = \begin{cases} -x & \text{if } x < 0.5, \\ 3\sin(10\pi x) & \text{if } 0.5 < x < 1.5, \\ -20(x-2)^2 & \text{if } 1.5 < x < 2.5, \\ 3 & \text{if } 2.5 < x, \end{cases} \tag{6.16}$$

which consists of jump discontinuities and smooth high-frequency oscillations. We discretize the domain $[0, 3]$ and generate a sequence of nested grids of the form (4.1) by setting $N_0 = 16$ and $K = 4$. We use the data on the grid $\mathcal{T}^k$, and interpolate it onto the grid $\mathcal{T}^{k+1}$ for $0 \leqslant k < K$. As shown in Figure 6.7, the interpolation results using ENO-p and the DeLENO-p classification network are identical on all grids for this particular function and $p = 3, 4$. For the DeLENO-3 regression network, the approximation results do not agree perfectly with the ENO-3 interpolation results. This however comes as no surprise given the discussion in Section 6.2.2 and Figure 6.4.

6.3.2 Data compression

A second application of ENO and DeLENO is data compression, in particular the compressed representation of functions. For this goal we introduce the multi-resolution representation of functions, with notation and operators similar to those introduced in [Aràndiga and Donat, 2000].

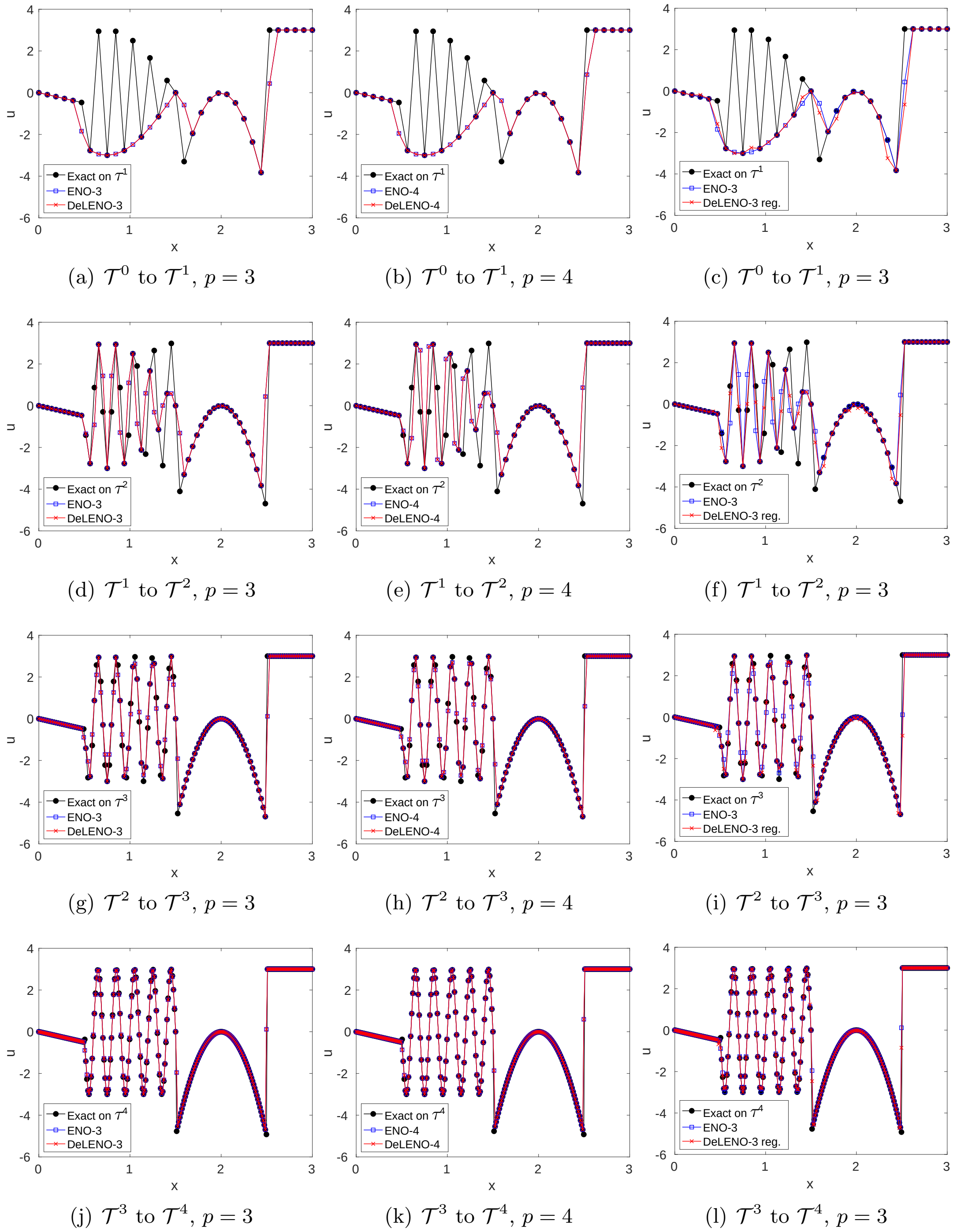
We define a sequence of nested uniform meshes $\{\mathcal{T}^k\}_{k=0}^K$ on $\Omega = [c, d]$, where

$$\mathcal{T}^k = \{I_i^k\}_{i=1}^{N_k}, \quad I_i^k = [x_{i-1}^k, x_i^k], \quad x_i^k = c + ih_k, \quad h_k = \frac{(d-c)}{N_k}, \quad N_k = 2^k N_0, \tag{6.17}$$

for $0 \leqslant i \leqslant N_k$, $0 \leqslant k \leqslant K$ and where N_0 is some positive integer. We call $\{x_i^k\}_{i=0}^{N_k}$ the nodes of the mesh $\mathcal{T}^k$. Let $\mathcal{B}_\Omega$ be the set of bounded functions on Ω and $\mathcal{V}^n$ the space of real-valued finite sequences of length n. Then we can define the following operators associated with the various meshes:

- The *discretizer* $D_k : \mathcal{B}_\Omega \mapsto \mathcal{V}^{N_k+1}$ defined by

$$D_k f = q^k := \{q_i^k\}_{i=0}^{N_k} = \{q(x_i^k)\}_{i=0}^{N_k}, \quad \forall\ q \in \mathcal{B}_\Omega.$$



(a) $\mathcal{T}^0$ to $\mathcal{T}^1$, $p = 3$

(b) $\mathcal{T}^0$ to $\mathcal{T}^1$, $p = 4$

(c) $\mathcal{T}^0$ to $\mathcal{T}^1$, $p = 3$

(d) $\mathcal{T}^1$ to $\mathcal{T}^2$, $p = 3$

(e) $\mathcal{T}^1$ to $\mathcal{T}^2$, $p = 4$

(f) $\mathcal{T}^1$ to $\mathcal{T}^2$, $p = 3$

(g) $\mathcal{T}^2$ to $\mathcal{T}^3$, $p = 3$

(h) $\mathcal{T}^2$ to $\mathcal{T}^3$, $p = 4$

(i) $\mathcal{T}^2$ to $\mathcal{T}^3$, $p = 3$

(j) $\mathcal{T}^3$ to $\mathcal{T}^4$, $p = 3$

(k) $\mathcal{T}^3$ to $\mathcal{T}^4$, $p = 4$

(l) $\mathcal{T}^3$ to $\mathcal{T}^4$, $p = 3$

Figure 6.7: Interpolating the function (6.16) using ENO-3 and the DeLENO-3 classification network (left), ENO-4 and the DeLENO-3 classification network (middle) and ENO-3 and the DeLENO-3 regression network (right).

- The *reconstructor* $R_k : \mathcal{V}^{N_k+1} \mapsto \mathcal{B}_\Omega$ satisfying $D_k R_k q^k = q^k$ for $q^k \in \mathcal{V}^{N_k+1}$. Thus, $(R_k q^k)(x)$ interpolates the members of q^k at the nodes of $\mathcal{T}^k$.
- The *decimator* $D_k^{k-1} : \mathcal{V}^{N_k+1} \mapsto \mathcal{V}^{N_{k-1}+1}$ defined by $D_k^{k-1} := D_{k-1} R_k$. For $q \in \mathcal{B}_\Omega$, we have

$$q_i^{k-1} = (D_k^{k-1} q^k)_i = q_{2i}^k, \quad 0 \leqslant i \leqslant N_{k-1}. \tag{6.18}$$

- The *predictor* $P_{k-1}^k : \mathcal{V}^{N_{k-1}+1} \mapsto \mathcal{V}^{N_k+1}$ defined by $P_{k-1}^k := D_k R_{k-1}$. The predictor tries to recover the function values q^k from the coarser data q^{k-1}, for $q \in \mathcal{B}_\Omega$.

The prediction error made by the predictor is given by

$$e_i^k = q_i^k - (P_{k-1}^k q^{k-1})_i, \quad 0 \leqslant i \leqslant N_k.$$

Clearly $e_{2i}^k = 0$ for $0 \leqslant i \leqslant N_{k-1} = N_k/2$. Thus, the interpolation error is essentially evaluated at the nodes in $\mathcal{T}^k \setminus \mathcal{T}^{k-1}$, which we denote as

$$d_i^k = e_{2i-1}^k = q_{2i-1}^k - (P_{k-1}^k q^{k-1})_{2i-1}, \quad 1 \leqslant i \leqslant N_{k-1}. \tag{6.19}$$

Given q^{k-1} and d^k, we can recover q^k using (6.18) and (6.19). By iteratively applying this procedure, the data q^k on the finest mesh can be fully encoded using the multiresolution representation

$$\{q^0, d^1, d^2, ..., d^K\}. \tag{6.20}$$

This multiresolution representation (6.20) for a function $f \in \mathcal{B}_\Omega$ is convenient to perform data compression. The easiest compression strategy [Aràndiga and Donat, 2000] corresponds to setting the coefficients d_i^k in (6.19) to zero based on a suitable threshold $\varepsilon^k \geqslant 0$:

$$\widehat{d}_i^k = \mathcal{G}(d_i^k; \varepsilon^k) = \begin{cases} 0 & \text{if } |d_i^k| \leqslant \varepsilon^k \\ d_i^k & \text{otherwise.} \end{cases} \tag{6.21}$$

When the thresholds are chosen to be of the form

$$\varepsilon^k = \varepsilon t^{K-k}, \quad 0 < t < 1, \tag{6.22}$$

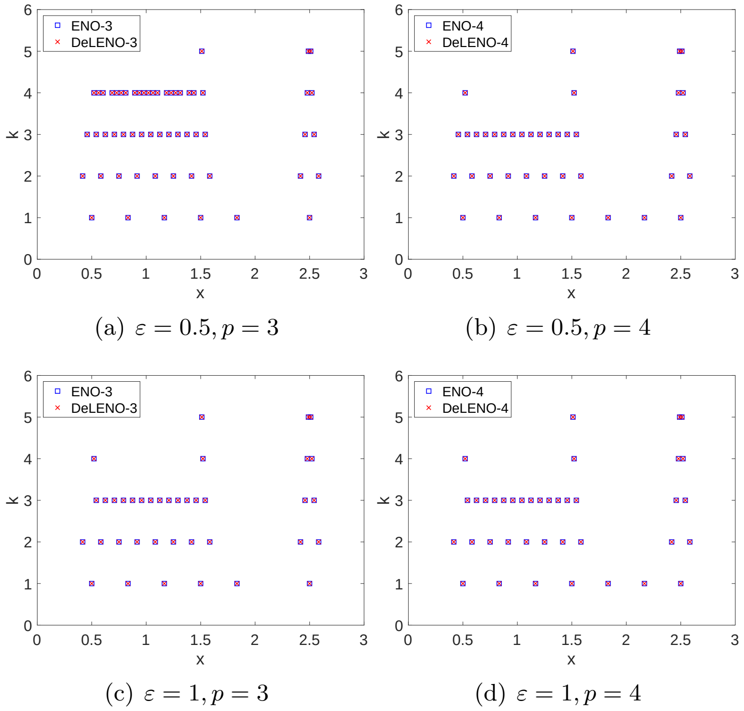
then Proposition B.1 provides an error bound for the compressed encoded representation of the form

$$\{f^0, \widehat{d}^1, \widehat{d}^2, ..., \widehat{d}^K\}. \tag{6.23}$$

The procedures for compressed encoding and decoding are listed in Algorithms 3 and 4 in Appendix B.

We now apply the multi-resolution representation framework of Appendix B to use DeLENO-p classification to compress the function $q = u$ from (6.16). We construct a nested sequence of meshes on $[0, 3]$ by choosing $N_0 = 9$ and $K = 5$ in (6.17). We use Algorithm 3 to obtain the multi-resolution representation of the form (6.23) and decode the solution using Algorithm 4 to obtain the approximation $\widehat{q}^K$. The compression thresholds needed for the encoding procedure are set using (6.22).

Figure 6.8 provides a comparison of the results obtained using different values for the threshold parameters ε, and shows the non-zero coefficients $\widehat{d}^k$ for each mesh level k. Note that a larger number of non-zero coefficients are required to represent the data in the high-frequency region (the graph of the function is shown in Figure 6.7). A higher value of ε can truncate a larger number of $\widehat{d}^k$ components, as is evident for $p = 3$. However, there is no qualitative difference between $\widehat{q}^K$ obtained for the two ε values considered. Thus, it is beneficial to use the larger ε, as it leads to a sparser multi-resolution representation without deteriorating the overall features. The solutions obtained with ENO and DeLENO are indistinguishable. We refer to Table 6.2 for the errors of the two methods.



(a) $\varepsilon = 0.5, p = 3$

(b) $\varepsilon = 0.5, p = 4$

(c) $\varepsilon = 1, p = 3$

(d) $\varepsilon = 1, p = 4$

Figure 6.8: Data compression of (6.16) using ENO-p and DeLENO-p classification networks with N_0, $L = 5$ and $t = 0.5$. The non-zero coefficients $\widehat{d}^k$ at each level (right).

p	ε	$\Vert q^K - \widehat{q}^K\Vert_1$		$\Vert q^K - \widehat{q}^K\Vert_2$		$\Vert q^K - \widehat{q}^K\Vert_\infty$	
		ENO	DeLENO	ENO	DeLENO	ENO	DeLENO
3	0.5	5.125e-2	5.125e-2	8.701e-2	8.701e-2	3.281e-1	3.281e-1
	1.0	2.072e-1	2.072e-1	2.421e-1	2.421e-1	4.102e-1	4.102e-1
4	0.5	1.032e-1	1.038e-1	1.268e-1	1.274e-1	3.027e-1	3.027e-1
	1.0	1.122e-1	1.122e-1	1.356e-1	1.356e-1	3.947e-1	3.947e-1

Table 6.2: 1D compression errors for (6.16).

The compression ideas used for one-dimensional problems can also be easily extended to handle functions defined on two-dimensional tensorized grids. We consider a sequence of grids $\mathcal{T}^k$ with $(N_k^x + 1) \times (N_k^y + 1)$ nodes, where $N_k^x = 2^k N_0^x$ and $N_k^y = 2^k N_0^x$, for $0 \leqslant k \leqslant K$. Let q^k be the data on grid $\mathcal{T}^k$ and denote by $\widehat{q}^{k+1}$ the compressed interpolation on grid $\mathcal{T}^{k+1}$. To obtain $\widehat{q}^{k+1}$, we first interpolate along the x-coordinate direction to obtain an intermediate approximation $\widetilde{q}^{k+1}$ of size $(N_{k+1}^x + 1) \times (N_k^y + 1)$.

Then we use $\widetilde{q}^{k+1}$ to interpolate along the y-coordinate direction to obtain the final approximation $\widehat{q}^{k+1}$.

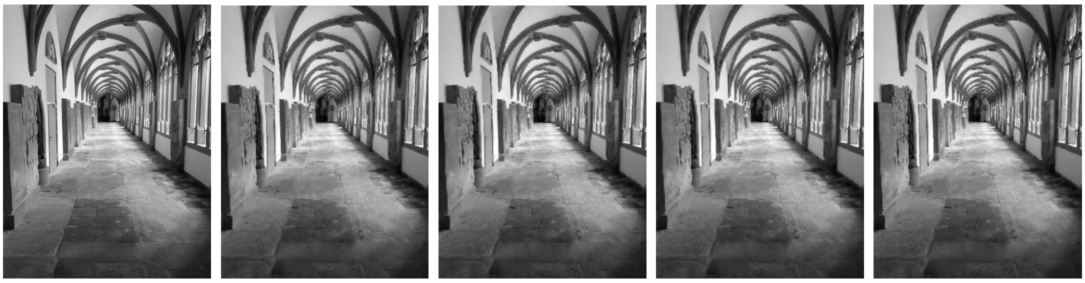
To illustrate this method, we use ENO and DeLENO classification networks to compress an image with 705 × 929 pixels, shown in Figure 6.9(a). We set $K = 5$, $\varepsilon = 1$, $t = 0.2$ in (6.22). Once again, ENO and DeLENO give similar results, as can be seen from the decompressed images in Figure 6.9 and the relative errors in Table 6.3. In this table we additionally listed the compression rate

$$c_r = 1 - \frac{\#\left\{d^k_{i,j} | d^k_{i,j} > \varepsilon^k,\ 1 \leqslant k \leqslant K\right\}}{(N^x_L + 1)(N^y_L + 1) - (N^x_0 + 1)(N^y_0 + 1)}, \tag{6.24}$$

which represents the fraction of coefficients set to zero.

p	Scheme	Rel. L^1	Rel. L^2	Rel. L^∞	c_r
3	ENO	5.346e-2	8.368e-2	5.194e-1	0.996
	DeLENO	5.343e-3	8.365e-2	5.194e-1	0.996
4	ENO	5.422e-2	8.485e-2	5.581e-1	0.996
	DeLENO	5.422e-2	8.492e-2	5.581e-1	0.996

Table 6.3: Image compression errors.



(a) Original (b) ENO-3 (c) DeLENO-3 (d) ENO-4 (e) DeLENO-4

Figure 6.9: Image compression.

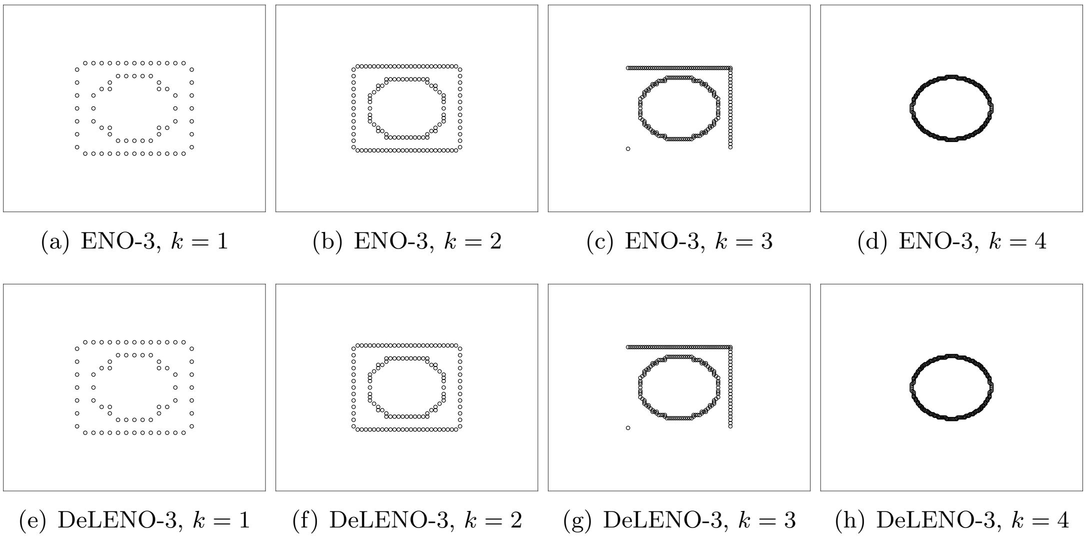
As an additional example of two-dimensional data compression, we consider the function

$$q(x, y) = \begin{cases} -10 & \text{if } (x - 0.5)^2 + (y - 0.5)^2 < 0.0225 \\ 30 & \text{if } |x - 0.5| > 0.8 \text{ or } |y - 0.5| > 0.8\ , \\ 40 & \text{otherwise} \end{cases} \tag{6.25}$$

where $(x, y) \in [0, 1] \times [0, 1]$, and generate a sequence of meshes by setting $K = 4$, $N^x_0 = 16$ and $N^y_0 = 16$. The threshold for data compression is chosen according to (6.22), with $\varepsilon = 10$ and $t = 0.5$. The non-zero $\widehat{d}^k$ coefficients are plotted in Figure 6.10, while the errors and compression rate (6.24) are listed in Table 6.4. Overall, ENO and DeLENO perform equally well, with DeLENO giving marginally smaller errors.

p	Scheme	Rel. L^1	Rel. L^2	Rel. L^∞	c_r
3	ENO	3.341e-3	2.442e-2	4.302e-1	0.989
	DeLENO	3.246e-3	2.367e-2	4.302e-1	0.989
4	ENO	3.816e-3	3.237e-2	5.876e-1	0.989
	DeLENO	3.681e-3	3.130e-2	5.876e-1	0.989

Table 6.4: 2D compression errors for (6.25).


(a) ENO-3, $k = 1$ (b) ENO-3, $k = 2$ (c) ENO-3, $k = 3$ (d) ENO-3, $k = 4$

(e) DeLENO-3, $k = 1$ (f) DeLENO-3, $k = 2$ (g) DeLENO-3, $k = 3$ (h) DeLENO-3, $k = 4$

Figure 6.10: Non-zero coefficients $\widehat{d}^k$ for data compression of (6.25) using ENO-3 and DeLENO-3 classification networks for mesh level $1 \leqslant k \leqslant 4$. The results for (DeL)ENO-4 are similar.

6.3.3 Conservation laws

Next, we present an example of how the ENO and DeLENO can be used for reconstruction purposes, when used to approximate solutions of conservation laws. We work in the framework of high-order finite difference schemes with flux-splitting and we use a fourth-order Runge-Kutta scheme for the time integration.

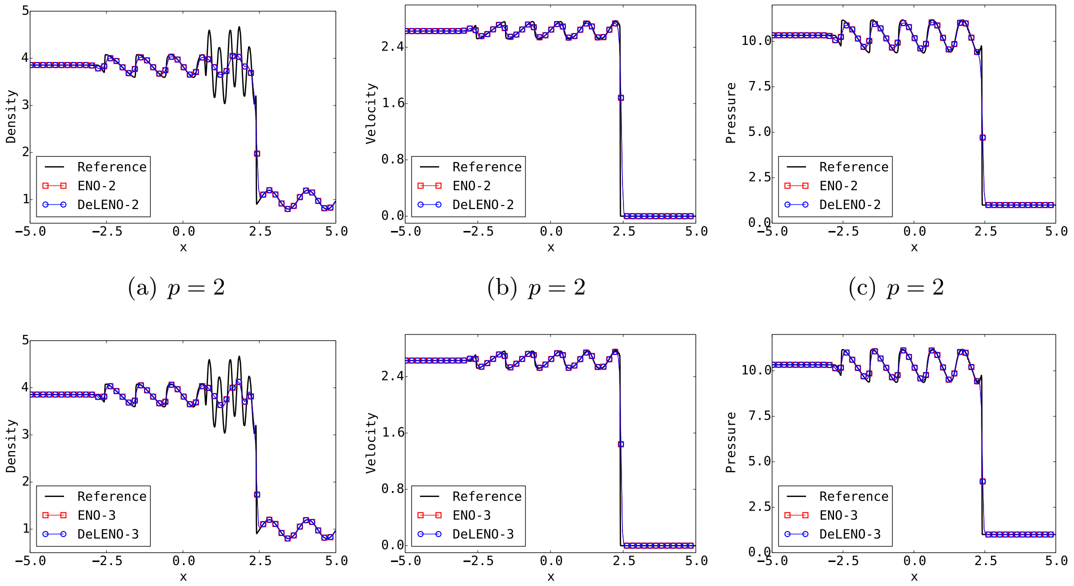
As a first example, we consider the system of conservation laws governing compressible flows given by

$$\partial_t \begin{pmatrix} \rho \\ v \\ p \end{pmatrix} + \partial_x \begin{pmatrix} \rho v \\ \rho v^2 + p \\ (E+p)v \end{pmatrix} = 0, \qquad E = \frac{1}{2}\rho v^2 + \frac{p}{\gamma - 1},$$

where ρ, v and p denote the fluid density, velocity and pressure, respectively. The quantity E represents the total energy per unit volume, where $\gamma = c_p/c_v$ is the ratio of specific heats, chosen as $\gamma = 1.4$ for our simulations. We consider the shock-entropy problem [Shu and Osher, 1989], which describes the interaction of a right moving shock with a smooth oscillatory waves. The initial conditions for this test case are prescribed as

$$(\rho,\ v,\ p) = \begin{cases} (3.857143,\ 2.629369,\ 10.33333) & \text{if } x < -4 \\ (1 + 0.2\sin(5x),\ 0,\ 1) & \text{if } x > -4 \end{cases},$$

on the domain $[-5,5]$. Due to the generation of high frequency physical waves, we solve the problem on a fine mesh with $N = 200$ cells up to $T_f = 1.8$ with CFL $= 0.5$. A reference solution is obtained with ENO-4 on a mesh with $N = 2000$ cells. As can be seen in Figure 6.11, ENO-p and the DeLENO-p classification networks perform equally well depending on the order p.



(a) $p = 2$ (b) $p = 2$ (c) $p = 2$

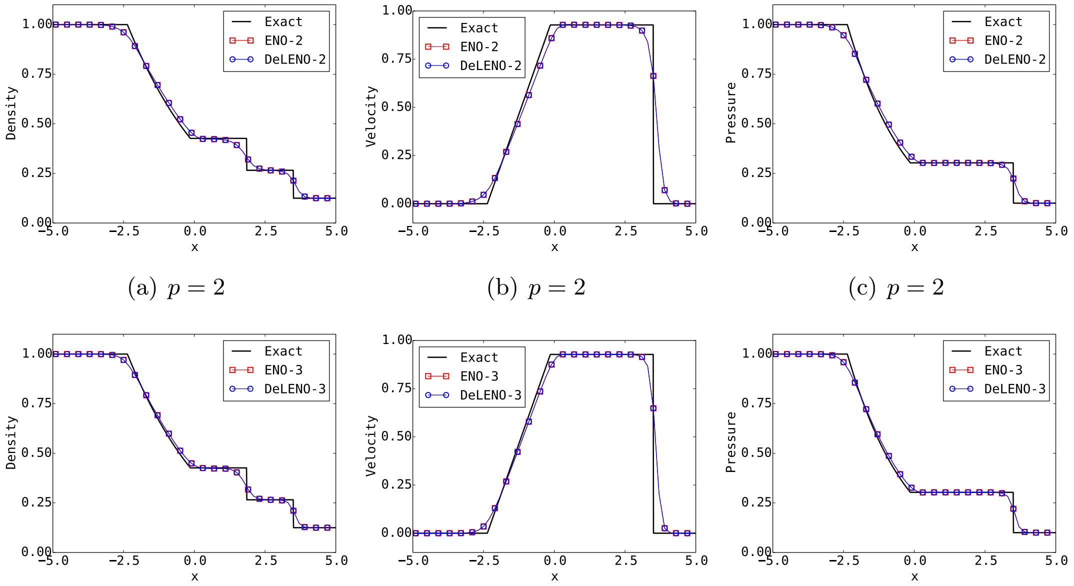
(d) $p = 3$ (e) $p = 3$ (f) $p = 3$

Figure 6.11: Solution for Euler shock-entropy problem with ENO-p and DeLENO-p on a mesh with $N = 200$ cells.

Next, we solve the Sod shock tube problem [Sod, 1978], whose initial conditions are given by

$$(\rho,\ v,\ p) = \begin{cases} (1,\ 0,\ 1) & \text{if } x < 0 \\ (0.125,\ 0,\ 0.1) & \text{if } x > 0 \end{cases},$$

on the domain $[-5,5]$. The solution consists of a shock wave, a contact discontinuity and a rarefaction. The mesh is descretized with $N = 50$ cells and the problem is solved until $T_f = 2$ with a CFL $= 0.5$. The solutions obtained with ENO-p and the DeLENO-p classification networks are identical, as depicted in Figure 6.12.



(a) $p = 2$

(b) $p = 2$

(c) $p = 2$

(d) $p = 3$

(e) $p = 3$

(f) $p = 3$

Figure 6.12: Solution for Euler Sod shock tube problem with ENO-p and DeLENO-p on a mesh with $N = 50$ cells.

Chapter 7

Conclusion

In this thesis, we considered the interpolation of rough functions, with special attention to piecewise smooth functions. Despite their very distinctive approaches, deep neural networks and the ENO(-SR) interpolation procedure are both suitable methods for the interpolation of such functions. Their efficiency respectively relies on the universality of deep neural networks and the data dependence of ENO. We argued that ENO interpolation is a classification problem at heart, where the classes correspond to stencil shifts, and subsequently proved the existence of a ReLU neural network that predicts the ENO stencil shift with an accuracy of 100%. The same thing was done for a new adaptation of ENO-SR that was proven to have the same order of accuracy. In addition, we constructed explicit ReLU neural networks that directly approximate the output of the ENO and ENO-SR interpolation procedures, proving error bounds where possible. These surprising results provide a different perspective on the ability of neural networks in approximating functions and reveal their enormous expressive power as even a highly non-linear, data dependent procedure such as ENO is nothing more than a ReLU neural network. By interpreting ENO as a neural network, we provide a natural framework for recasting the problem of interpolation in terms of pre-trained neural networks such as DeLENO, where the input vector of sample values is transformed by the network into the output vector of interpolated values. Thus, these networks are trained once and do not need to retrained for every new underlying function, in contrast to the methods of [Yarotsky, 2017], which may therefore have only limited utility in practice.

We trained both classification and regression networks, using the architectures from our theoretical results. The trained classification networks are able to predict the correct stencil shift with an accuracy of near 100% and therefore agree with the original ENO(-SR) procedure almost perfectly. On the other hand, the accuracy of the regression networks was seen to be highly dependent on the nature and size of the training data set, the network architecture and the chosen optimization algorithm. In particular, ENO's higher-order accuracy for smooth functions and its ability to interpolate rough functions without creating spurious oscillations were hard to reconcile during the training. Even when successful, a decay in the order of accuracy for smooth functions on fine grids seems unavoidable. Depending on the scaling procedure, the higher-order accuracy reduces to either first-order accuracy or a constant error. Nevertheless, the networks proved their use in a plethora of applications, notably interpolation, data compression and the approximation of solutions of nonlinear conservation laws.

Appendix A

Weights and biases for trained networks

We list the obtained weights and biases for the trained third-order DeLENO classification network, obtained using the training procedure of Section 6.1. The theoretical counterparts can be found in (4.19) and (4.20).

$$W^1 = \begin{pmatrix} 1.1951 & 2.0433 & -11.7410 & 5.6383 \\ 2.9216 & -2.8703 & -2.5077 & 2.4624 \\ -2.2775 & 7.6890 & -7.2667 & 2.4914 \\ 3.2909 & -5.8431 & -5.6085 & 3.4171 \end{pmatrix}, \quad b^1 = \begin{pmatrix} -0.1069 \\ -0.3615 \\ 0.0389 \\ 0.0605 \end{pmatrix},$$

$$W^2 = \begin{pmatrix} -11.6122 & 4.2986 & 10.7356 & 8.1240 \\ 11.5929 & -4.2767 & -10.7357 & -8.1316 \end{pmatrix}, \quad b^2 = \begin{pmatrix} -2.5193 \\ 2.4493 \end{pmatrix}.$$

Below we list the obtained weights and biases for the trained fourth-order DeLENO classification network. The theoretical counterparts can be found in (4.21-4.24).

$$W^1 = \begin{pmatrix} -0.0559 & -1.0026 & 1.1115 & 1.001 & 1 - 1.0569 & -0.0001 \\ -0.3547 & 0.3557 & -0.8777 & 2.0707 & -1.1983 & -0.0051 \\ -0.0060 & -0.6155 & 1.6342 & -1.4526 & 0.4599 & -0.0370 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0011 & -0.1965 & 0.7964 & -1.1817 & 0.7805 & -0.1913 \\ -0.3324 & 1.2088 & -1.6946 & 1.0632 & -0.2479 & -0.0043 \\ 0.1432 & -0.6459 & 0.9448 & -0.5434 & 0.1271 & -0.0154 \\ -0.0076 & -0.4239 & 0.8604 & -0.0248 & -0.8755 & 0.4517 \\ 0.0196 & -0.1527 & 0.6286 & -0.9194 & 0.6069 & -0.1619 \\ -0.0088 & -0.2571 & 1.0627 & -1.6288 & 1.0899 & -0.2714 \end{pmatrix}, b^1 = \begin{pmatrix} -0.0005 \\ 0.0029 \\ -0.0005 \\ -0.1217 \\ 0.0012 \\ 0.0007 \\ -0.0010 \\ 0.0010 \\ 0.0027 \\ -0.0005 \end{pmatrix},$$

$$W^2 = \begin{pmatrix} 0 & 1.6803 & 0.2910 & -3.1738 & 0 & 1.5946 \\ 0 & -1.9935 & -0.7975 & 3.2015 & 0 & -1.9124 \\ 0 & -0.3125 & 2.7085 & 0.7264 & 0 & -0.2819 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.3886 & 0.6976 & 0.6071 & 0 & -1.3514 \\ 0 & 2.1097 & 0.8511 & -3.6655 & 0 & 2.0568 \\ 0 & 0.6645 & 0.1485 & -1.2784 & 0 & 0.5052 \\ 0 & -0.0061 & -1.4376 & -0.0073 & 0 & -0.0082 \\ 0 & 0.4580 & -1.0432 & 0.0474 & 0 & 0.4542 \\ 0 & 0.6597 & -2.6379 & -0.4588 & 0 & 0.7357 \end{pmatrix}^T, \quad b^2 = \begin{pmatrix} -0.0349 \\ 0.0993 \\ 0.0463 \\ 0.0284 \\ -0.0570 \\ 0.0438 \end{pmatrix},$$

$$W^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8.0933 & 0.6463 & -5.7734 & 0 & 8.3502 \\ 0 & 2.0780 & -10.0148 & -0.3565 & 0 & 1.8241 \end{pmatrix}, \quad b^2 = \begin{pmatrix} -0.0316 \\ -0.0432 \\ -0.3800 \\ 0.4131 \end{pmatrix},$$

$$W^4 = \begin{pmatrix} 0 & 0 & 2.3377 & -11.5452 \\ 0 & 0 & 2.2965 & 11.1520 \\ 0 & 0 & -12.6485 & -0.8555 \end{pmatrix}, \quad b^4 = \begin{pmatrix} 1.6841 \\ -7.5807 \\ 8.2575 \end{pmatrix}.$$

Appendix B

Multi-resolution representation of functions for data compression

In Section 6.3.2, the multi-resolution representation of functions was introduced to provide a framework for data compression of one-dimensional and two-dimensional functions. Below we list the algorithms to obtain the compressed representation of a function as in (6.23) (Algorithm 3) and to decode this compressed representation again (Algorithm 4).

Algorithm 3 Compressed encoding [Aràndiga and Donat, 2000]

```
Input: Highest resolution data $f^K$, number of levels $K$, number of points $N_0$ on coarsest mesh, ENO order $p$, threshold parameters $\varepsilon$ and $t$.
Output: Multi-resolution representation $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$.
  for $k = K$ to 1 do
    $f^{k-1} = D_k^{k-1} f^k$
  end for
  $\hat{f}^0 = f^0$
  for $k = 1$ to $K$ do
    $\widehat{f}^k_0 = f^K_0$
    Construct $P^k_{k-1}$ using Algorithm 1 and (4.13)
    $\widetilde{f}^k = P^k_{k-1} \widehat{f}^{k-1}$
    $N = N_0 2^{k-1}$
    for $i = 1$ to $N$ do
      $d^k_i = f^k_{2i-1} - \widetilde{f}^k_{2i-1}$
      $\varepsilon^k = \varepsilon t^{K-k}$
      $\widehat{d}^k_i = \mathcal{G}(d^k_i; \varepsilon^k)$
      $\widehat{f}^k_{2i-1} = \widetilde{f}^k_{2i-1} + \widehat{d}^k_i$
      $\widehat{f}^k_{2i} = \widehat{f}^{k-1}_i$
    end for
  end for
  return $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$
```

Algorithm 4 Decoding multi-resolution data [Aràndiga and Donat, 2000]

```
Input: Multi-resolution representation $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$, number of levels $K$, number of
  cells $N_0$ on coarsest mesh, ENO order $p$.
Output: Decoded function $\widehat{f}^K$.
  $\hat{f}^0 = f^0$
  for $k = 1$ to $K$ do
    Construct $P^k_{k-1}$ using Algorithm 1 and (4.13)
    $\widehat{f}^k = P^k_{k-1}\widehat{f}^{k-1} + \widehat{d}^k$
  end for
  return $\widehat{f}^K$
```

In addition, we present the following result on the error bounds for the compressed encoding in the form (6.23), the proof can be found in [Aràndiga and Donat, 2000].

Proposition B.1 *Let $\{\Omega^k\}_{l=0}^K$ be a sequence of nested uniform meshes discretizing the interval $[c, d]$ generated according to (6.17) for some positive integer $N_0 > 1$. Assume that some $f \in \mathcal{B}[c, d]$ is encoded using thresholds*

$$\varepsilon^k = \varepsilon t^{K-k}, \quad 0 < t < 1. \tag{B.1}$$

to give rise to the multi-resolution representation of the form (6.23). If $\widehat{f}^K$ is the decoded data, then

$$\|f^K - \widehat{f}^K\|_n \leqslant C_n\varepsilon \quad \textit{for } n = \infty, 1, 2, \tag{B.2}$$

where $C_\infty = (1-t)^{-1}$, $C_1 = (b-a)(1-t)^{-1}$ and $C_2 = \sqrt{(b-a)(1-t^2)^{-1}}$. This estimate is independent of the interpolation procedure used to encode and decode the data.

Bibliography

[Aràndiga et al., 2005] Aràndiga, F., Cohen, A., Donat, R., and Dyn, N. (2005). Interpolation and approximation of piecewise smooth functions. *SIAM Journal on Numerical Analysis*, 43(1):41–57.

[Aràndiga and Donat, 2000] Aràndiga, F. and Donat, R. (2000). Nonlinear multiscale decompositions: The approach of a. harten. *Numerical Algorithms*, 23(2):175–216.

[Barron, 1993] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

[Dafermos, 2010] Dafermos, C. M. (2010). *Hyperbolic conservation laws in continuum physics*, volume 325 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, third edition.

[De Ryck et al., 2020] De Ryck, T., Mishra, S., and Ray, D. (2020). On the approximation of rough functions with deep neural networks. Technical Report 2020-07, Seminar for Applied Mathematics, ETH Zürich, Switzerland.

[Evans, 1998] Evans, L. C. (1998). *Partial Differential Equations.* American Mathematical Society.

[Eyink and Sreenivasan, 2006] Eyink, G. L. and Sreenivasan, K. (2006). Onsager and the theory of hydrodynamic turbulence. *Rev. Modern Phys.*, 78(1):87–135.

[Fjordholm et al., 2013] Fjordholm, U. S., Mishra, S., and Tadmor, E. (2013). Eno recontruction and eno interpolation are stable. *Found. Comp. Math*, 13(2):139–159.

[Funahashi, 1989] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Grohs et al., 2019a] Grohs, P., Hornung, F., Jentzen, A., and Zimmermann, P. (2019a). Space-time error estimates for deep neural network approximations for differential equations.

[Grohs et al., 2019b] Grohs, P., Perekrestenko, D., Elbrächter, D., and Bölcskei, H. (2019b). Deep neural network approximation theory. *IEEE Transactions on Information Theory.*

[Hanin and Sellke, 2017] Hanin, B. and Sellke, M. (2017). Approximating continuous functions by ReLU nets of minimal width.

[Harten, 1986] Harten, A. (1986). On high-order accurate interpolation for non-oscillatory shock capturing schemes. *The IMA Volumes in Mathematics and Its Applications Oscillation Theory, Computation, and Methods of Compensated Compactness*, page 71–105.

[Harten, 1989] Harten, A. (1989). ENO schemes with subcell resolution. *Journal of Computational Physics*, 83(1):148–184.

[Harten et al., 1987] Harten, A., Engquist, B., Osher, S., and Chakravarthy, S. R. (1987). Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 71(2):231–303.

[Harten et al., 1997] Harten, A., Engquist, B., Osher, S., and Chakravarthy, S. R. (1997). Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 131(1):3 – 47.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Laney, 2007] Laney, C. B. (2007). *Computational gasdynamics.* Cambridge Univ. P.

[Leshno et al., 1993] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867.

[Lu et al., 2017] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc.

[Lye et al., 2019] Lye, K. O., Mishra, S., and Ray, D. (2019). Deep learning observables in computational fluid dynamics.

[Rolnick and Tegmark, 2017] Rolnick, D. and Tegmark, M. (2017). The power of deeper networks for expressing natural functions.

[Shu, 1998] Shu, C.-W. (1998). *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws*, pages 325–432. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Shu and Osher, 1989] Shu, C.-W. and Osher, S. (1989). Efficient implementation of essentially non-oscillatory shock-capturing schemes, ii. *Journal of Computational Physics*, 83(1):32 – 78.

[Shu and Osher, 1991] Shu, C. W. and Osher, S. (1991). High-order essentially nonoscillatory schemes for hamilton-jacobi equations. *SIAM J. Num. Anal.*, 28(4):107–122.

[Sod, 1978] Sod, G. A. (1978). A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1 – 31.

[Yarotsky, 2017] Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103 – 114.

[Yarotsky, 2018a] Yarotsky, D. (2018a). Optimal approximation of continuous functions by very deep ReLU networks.

[Yarotsky, 2018b] Yarotsky, D. (2018b). Universal approximations of invariant maps by neural networks.

[Yarotsky and Zhevnerchuk, 2019] Yarotsky, D. and Zhevnerchuk, A. (2019). The phase diagram of approximation rates for deep neural networks.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

On the Approximation of Rough Functions with Artificial Neural Networks

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
De Ryck	Tim

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the '<u>Citation etiquette</u>' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date
Zürich, 31/01/2020

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.